A Parallel Algorithm Using Perlin Noise Superposition Method for Terrain Generation Based on CUDA architecture

Huailiang Li [1,a], Xianguo Tuo [1,2], Yao Liu[1], Xin Jiang [2]

[1]Fundamental Science on Nuclear Wastes and Environmental Safety Laboratory ,Southwest University of Science and Technology, Mianyang 621010, China ;

[2]State Key Laboratory of Geohazard Prevention & Geoenvironmental Protection, Chengdu University of Technology, Chengdu 610059, China

[a]li-huai-liang@163.com

Keywords: Perlin Noise, CUDA, GPU, Terrain generation

Abstract. A parallel algorithm for terrain generation based on CUDA architecture is proposed in this paper, which aims to address the problems of high computational load and low efficiency when generating large scale terrains using the Perlin noise superposition method. The Perlin noise superposition method is combined with independent calculation of each point based on the characteristics of all adjacent points. The Perlin noise value of each terrain grid point is transferred to a GPU thread for calculation, so that the terrain generation process is executed in completely parallel in the GPU. Experimental results show that The GPU algorithm generates a grid of size 25000000 (25 million grid points) needs only 0.6355 s, while the original CPU algorithm takes 23.3723 s, so, the parallel processing algorithm can improve the efficiency of the terrain generation and meet the requirements for large-scale terrain generation compared with the original algorithm.

Introduction

Three-dimensional terrain is one of the most fundamental and important technology in the development visualization system[1], and widely applied in the area of Geographic Information System(GIS), Virtual Reality(VR), Synthetic Natural Environment(SNE), flight simulators, and even in the Vehicle Terrain Measurement System(VTMS)[2-4]. At present, a simulation method based on the digital elevation model (DEM) and terrain generation algorithms based on fractal theory is commonly used to generate terrain [5,6], such as the random midpoint displacement method[7,8], Perlin noise superposition method[9,10], and so on. The Perlin noise function was proposed by Ken Perlin in 1985[11] and was improved in 2002[12]. The Perlin noise has been used in many fields, such as generation of emotional expressions in a robot[13-15], traffic generator[16], simulation of oxide textures[17], simulation of breast tissue[18], music generation system[19], and generation of procedural terrain[9,10,20]. The Perlin noise superposition, is widely used as that it can flexibly generate different terrains to meet various needs. However, there are some huge domains, such as catchments, mountains or gorges, where an accurate digital terrain model is required to mimic the complex topography of the terrain[21]. It means a large amount of data, complicated computing processes, and a longer execution time.

To address this problem, a parallel algorithm for terrain generation using the Perlin noise superposition method based on Compute Unified Device Architecture(CUDA) architecture was proposed in this paper. Initialization of the gradient search sequence is generated with the CPU and the Perlin noise value of each terrain mesh point is calculated within a thread that is executed in the GPU. Therefore, the terrain height value calculation is completely accomplished in the GPU which significantly improves the speed of terrain data generation. The Perlin noise terrain generation method is explained in further detail in the next section. The third section describes the terrain generation process of the parallel algorithm. In the fourth section, the results of experiments are analyzed to compare the efficiency of the two different algorithms to generate terrains of different scales. The fifth section generalizes and summarizes the work presented in this paper and finally draws a conclusion.

Perlin noise terrain generation method

The algorithm is based on the fact that the height of each terrain grid point can be generated using superposition of different frequencies and amplitudes of the Perlin noise value. A modified process of the Perlin noise terrain generation algorithm is demonstrated as follows.

A. Generate a sequence of positive integers: S (1~n) [in this article, n=256. $S_i = i(i = 1,2,...,n)$],

B. Find the sequence G according to the S gradient generation, $L_G = 2n$,

First, de-serialize the sequence using the random exchange method, i.e. choosing any random index location in the sequence between the initial position and the end position and then exchange the values: $S_i \leftrightarrow S_j \{ j = [1 + n \times rand()/RADN\_MAX] \ i = 1,2,...,n \}$.

Then, G can be generated from the de-serialized sequence S, as shown in Equation (1)

$$G_i = G_{i+n} = S_i \tag{1}$$

C. Set point $P'(x', y', z')$, $x', y', z' \in R^+$, $freq > 0$, $amp > 0$. Calculating the Perlin noise value of this point.

① Get $P(x, y, z)$ from point P using Equation (2):

$$\begin{cases} x = x' \bullet freq - [x' \bullet freq] \\ y = y' \bullet freq - [y' \bullet freq] \\ z = z' \bullet freq - [z' \bullet freq] \end{cases} \tag{2}$$

② Get the gradient value of 8 different points $P_i(x_i, y_i, z_i)$ near point P using the sequence G. $i \in \{1,2,3,...,8\}$, $\alpha = -1$, $x_i \in \{x, x+\alpha\}$, $y_i \in \{y, y+\alpha\}$, $z_i \in \{z, z+\alpha\}$

$$\text{Order} \begin{cases} X = [x'] \wedge (n-1) \\ Y = [y'] \wedge (n-1) \\ Z = [z'] \wedge (n-1) \\ f(i) = G_i, i = 1,2,3...,2n \end{cases} \tag{3}$$

Then, the search value m of point P_i can be determined using the following formula:

$$m = f(f(f(X) + Y) + Z) \tag{4}$$

Order $h = m \wedge 15$. Then, the gradient value of this point can be calculated using Equation (5).

$$\begin{aligned} g_i = & \left\{ \begin{array}{l} \dfrac{\max\{7-h,-1\}+1}{|7-h|+1} \bullet (-1)^h + \\ \min\{(h-13)(h-15),1\} \bullet \dfrac{\max\{h-12,-1\}+1}{|h-12|+1} \bullet (-1)^{(4+[\frac{h}{2}])} \end{array} \right\} \bullet x_i \\ & + \left\{ \frac{\max\{3-h,-1\}+1}{|3-h|+1} \bullet (-1)^{[\frac{h}{2}]} + \frac{\max\{h-8,-1\}+1}{|h-8|+1} \bullet (-1)^h \right\} \bullet y_i \\ & + \left\{ \min\{(h-12)(h-14),1\} \bullet \frac{\max\{h-4,-1\}+1}{|h-4|+1} \bullet (-1)^{(4+[\frac{h}{2}])} \right\} \bullet z_i \end{aligned} \tag{5}$$

$i = 1,2,3,...,8$, $\alpha = 1$, $x_i \in \{x, x+\alpha\}$, $y_i \in \{y, y+\alpha\}$, $z_i \in \{z, z+\alpha\}$.

The corresponding relationship between an adjacent point P_i and the gradient value g_i is as shown in Table 1.

Tab.1 Adjacent points and their corresponding gradients

i	Pi	gi
1	(x, y, z)	g_1
2	(x-1, y, z)	g_2
3	(x, y-1, z)	g_3
4	(x-1, y-1, z)	g_4
5	(x, y, z-1)	g_5
6	(x-1, y, z-1)	g_6
7	(x, y-1, z-1)	g_7
8	(x-1, y-1, z-1)	g_8

③ First, get smooth values of x, y and z using a smoothing function, as shown in formula(6),

$$\begin{cases} u = 6x^5 - 15x^4 + 10x^3 \\ v = 6y^5 - 15y^4 + 10y^3 \\ w = 6z^5 - 15z^4 + 10z^3 \end{cases} \tag{6}$$

Then, get the Perlin noise Lz using the following interpolation method, which uses an interpolation function to get the eight gradient values and the smooth values u, v and w.

$$\begin{cases} Lx_1 = g_1 + u \bullet (g_2 - g_1) \\ Lx_2 = g_3 + u \bullet (g_4 - g_3) \\ Lx_3 = g_5 + u \bullet (g_6 - g_5) \\ Lx_4 = g_7 + u \bullet (g_8 - g_7) \\ Ly_1 = Lx_1 + v \bullet (Lx_2 - Lx_1) \\ Ly_2 = Lx_3 + v \bullet (Lx_4 - Lx_3) \\ Lz = Ly_1 + w \bullet (Ly_2 - Ly_1) \end{cases} \tag{7}$$

D. Set $Octave \in N^+$, then get the Perlin noise value as the sum of points P. The evaluation process is as follows:

Order freq = 4, amp =1, octaves= 8; sum = 0;

While octaves > 0

sum = sum + Lz

freq = freq * 2

amp = amp * 1/2

octaves = octaves - 1

End

Figure 1 shows that the generated terrain through rendering data resulting from step (4) (100 * 100 mesh).

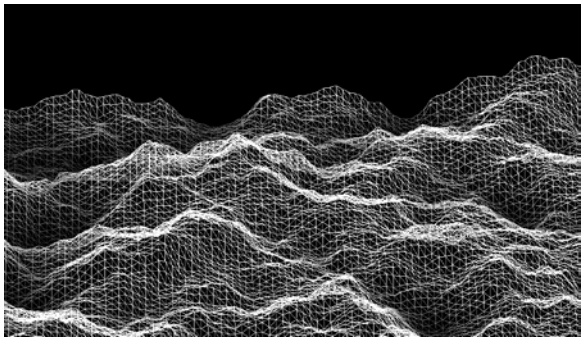

Fig.1 The graph of terrain generated by Perlin noise superposition (100*100)

Terrain generation parallel algorithm

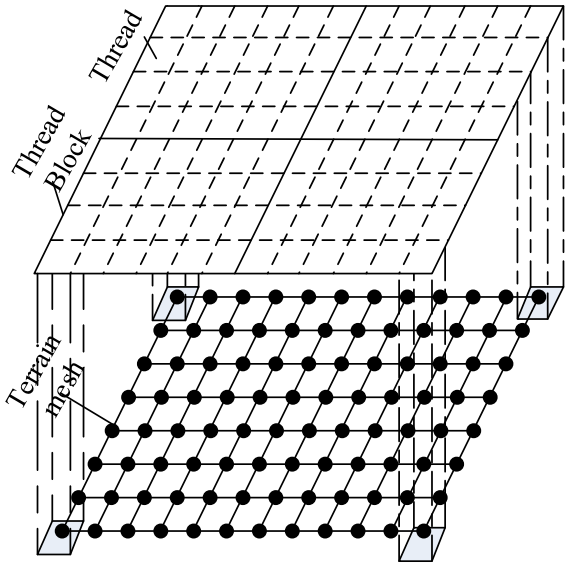

Fig.2 Graph of the GPU thread and its corresponding terrain of grid points

In the original algorithm, the height value of each point is obtained by calculating the Perlin noise value of that point, and calculation of each point is not related to calculation of other adjacent points. The parallel algorithm method described in this paper assigns each thread in the CUDA thread block to a terrain of grid points, as shown in Figure 2, and then calculates the Perlin noise value of this point in each thread. The final result will be written back to the CPU to be saved.

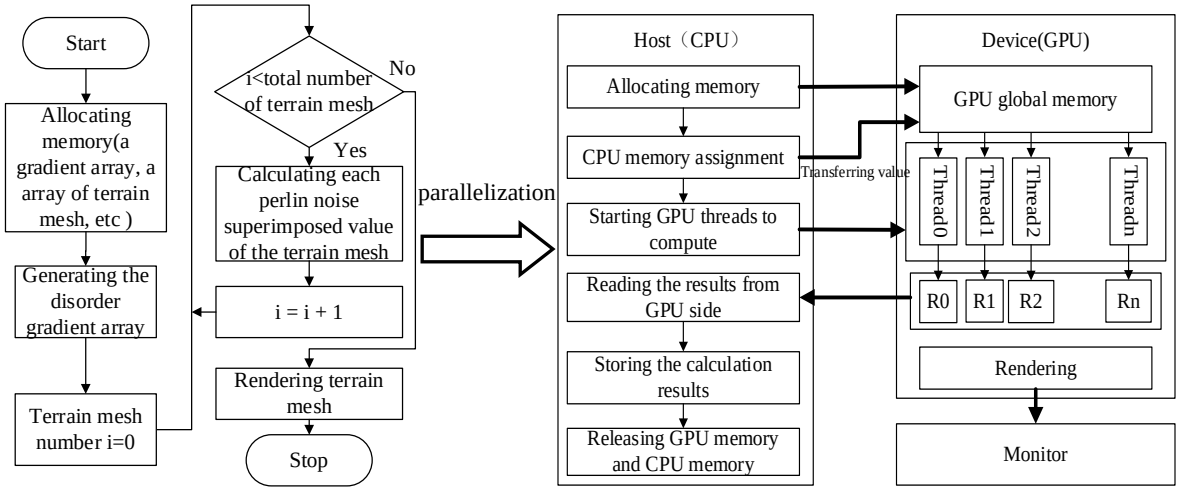

Fig.3The graph of CPU and GPU function module

The parallel algorithm requires that the CPU and GPU must work jointly, and they are responsible for different functional areas. As shown in Fig.3, controlling operations are done by CPU, and a large number of calculations are completed through the multi-threaded parallel computing work with GPU. The specific process description of the parallel algorithm is as follows:

CPU end application memory and its initialization data	
Step1	Allocate CPU memory: terrain mesh point height value array data[row * column] (row * column for terrain mesh point total) Allocate GPU end memory: gradient array dev_G [2n] (n = 256), with the GPU end terrain data corresponding to the grid array dev_data [row * column]
Step2	Generate a sequence of positive integers: S (1~n). ($n = 256$)
Step3	Determine sequence G based on the S gradient generation, using the method in 2 (2) $L_G = 2n$ and write the result in dev_G[2n]
Step4	Invoke the GPU end kernel function. Start threads with a total number of blockNum * threadNum = row *, each corresponding to a grid point, and then calculate the Perlin noise value of each point.
Step5	dev_data $\rightarrow$ data Write the results back to the CPU and release the memory for the data e.g. dev_data, dev_G, etc.
Calculate Perlin noise value of the corresponding terrain mesh point through GPU execution	
Step1	Get the present thread tid = threadIdx.x +blockIdx.x*blockDim
Step2	If tid > blockNum*threadNum, then back or execute step3
Step3	Execute 2(4), save the calculation results in dev_data[tid] i.e. dev_data[tid]=sum.

Experimental results and analysis

Thirty experiments to generate different sizes of grid terrains were carried out on both the GPU Parallel algorithm and the CPU original algorithm. The experimental hardware environment was as follows: the processor used was Intel (R) Core (TM) i5-3470, with 3.2GHz CPU and 3.47 GB of memory, the graphics card was NVIDIA GeForce GT620 and the software environment: QT/C + + and CUDA/C.

The average time taken to generate different grid sizes using the two algorithms is presented in Table 2. This data was used to generate the elapsed time contrast diagram which is shown in Figure 4. This graph shows that when the local grid scale is small, the GPU algorithm is more time-consuming than the CPU algorithm. This is mainly because most of the time is spent doing the data transfer between the CPU and GPU although less time is taken for the calculation. The original algorithm can be done rapidly by the CPU and the no benefit from parallel computing is observed. However, as the grid scale increases, the time taken to complete the original algorithm grows approximately exponentially, while the time taken to complete the parallel algorithm grows approximately linearly. The GPU algorithm generates a grid of size 25000000 (25 million grid points) in 0.6355 s, while the original CPU algorithm takes 23.3723 s. Therefore, the GPU parallel algorithm is more efficient than the original CPU algorithm for generation of large scale terrain meshes.

Tab.2 Time comparison for generation of different mesh sizes using the CPU and GPU algorithms

Terrain grid number(n)	$\log_{10}(n)$	Time Taken-CPU(s)	Time Taken-GPU(s)
40000	4.6021	0.0369	0.0517
160000	5.2041	0.1482	0.0559
360000	5.5563	0.3340	0.0606
640000	5.8062	0.5929	0.0689
1000000	6.0000	0.9287	0.0770
4000000	6.6021	3.7359	0.1529
9000000	6.9542	8.3634	0.2691
16000000	7.2041	14.8535	0.4227
25000000	7.3979	23.3723	0.6355

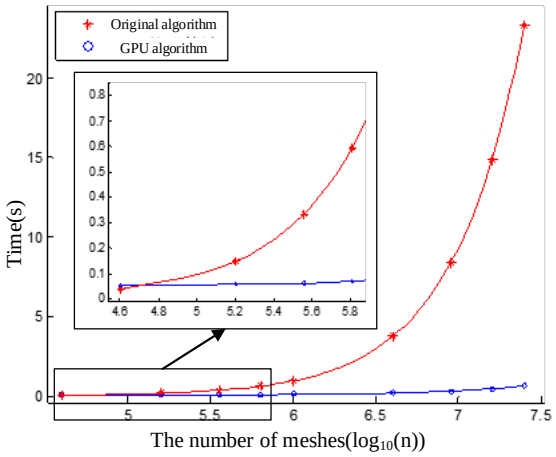

Fig.4 Graph showing time comparison of the two algorithms

Conclusion

This article firstly introduced an algorithm to generate terrain using Perlin noise. A terrain generation parallel algorithm based on CUDA architecture was then proposed which combined the characteristics of the original algorithm with parallel processing. The height value of each point in the terrain grid is transferred to the corresponding GPU parallel thread where it is executed. Comparison of the times taken to generate different sizes of terrain grid using both algorithms shows that the proposed algorithm achieves much faster speed at larger terrain sizes, which satisfies requirements for generating massive terrains.

Acknowledgement

This research was supported by National Natural Science Foundation of China (NSFC) program No. 41227802; Sichuan Province Science Supporting Program Foundation (No. 2014GZ0184); and by Research Funding of Southwest University of Science and Technology (No. 13zx7135, 14tdhk03 and 15yyhk14).

References

[1] Bi W, Zang W, Liu T. Three-Dimensional Terrain Modeling and Path Optimization on it Based on Google Earth and ACIS[M]. Human Centered Computing. Springer International Publishing, 2015: 872-879. doi: 10.1007/978-3-319-15554-8_81.

[2] Wang H, Zhang L, Mai J, et al. Spatial Multi-resolution Terrain Rendering Based on the Improved Strategy for Terrain Dispatching and Pre-reading[M]. Geo-Informatics in Resource Management and Sustainable Ecosystem. Springer Berlin Heidelberg, 2015: 225-234. doi: 10.1007/s00371-014-0941-6.

[3] Wang H, Mai J, Song Y, et al. A 3D visualization framework for real-time distribution and situation forecast of atmospheric chemical pollution[M]. AsiaSim 2013. Springer Berlin Heidelberg, 2013: 415-420. doi: 10.1007/978-3-642-45037-2_44.

[4] Chung H, North C, Ferris J. Developing Large High-Resolution Display Visualizations of High-Fidelity Terrain Data[J]. Journal of Computing and Information Science in Engineering, 2013, 13(3): 034502. doi: 10.1115/1.4024656.

[5] Lee C, Oh J, Hong C, et al. Automated Generation of a Digital Elevation Model Over Steep Terrain in Antarctica From High-Resolution Satellite Imagery[J]. Geoscience and Remote Sensing,

IEEE Transactions on, 2015, 53(3): 1186-1194. doi: 10.1109/TGRS.2014.2335773.

[6] Noh M J, Howat I M. Automated stereo-photogrammetric DEM generation at high latitudes: Surface Extraction with TIN-based Search-space Minimization (SETSM) validation and demonstration over glaciated regions[J]. GIScience & Remote Sensing, 2015, 52(2): 198-217. doi: 10.1080/15481603.2015.1008621.

[7] Zhang X, Yuan Y, Qi M. Impact of terrain complexity on the accuracy of calculations of river channel storage volume derived from measurements of underwater topography[J]. Arabian Journal of Geosciences, 2015: 1-20. doi: 10.1080/15481603.2015.1008621.

[8] Génevaux J D, Galin E, Peytavie A, et al. Terrain Modelling from Feature Primitives[C]. Computer Graphics Forum. 2015. doi: 10.1111/cgf.12530.

[9] Michelon de Carli D, Pozzer C T, Bevilacqua F, et al. Procedural generation of 3D canyons[C]. Graphics, Patterns and Images (SIBGRAPI), 2014 27th SIBGRAPI Conference on. IEEE, 2014: 103-110. doi: 10.1109/SIBGRAPI.2014.41.

[10] Choroś K, Topolski J. A Method of the Dynamic Generation of an Infinite Terrain in a Virtual 3D Space[M]. Intelligent Information and Database Systems. Springer International Publishing, 2015: 377-387. doi: 10.1007/978-3-319-15705-4_37.

[11] Perlin.K. An Image Synthesizer[J]. Computer Graphics. 1985, 19(3). doi: 10.1145/325165.325247.

[12] Perlin.K. Improving Noise[J]. Computer Graphics, 2002, 35(3).doi: 10.1145/566654.566636.

[13] Andrist S, Tan X Z, Gleicher M, et al. Conversational gaze aversion for humanlike robots[C]. Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction. ACM, 2014: 25-32. doi: 10.1145/2559636.2559666.

[14] Bohg J, Romero J, Herzog A, et al. Robot arm pose estimation through pixel-wise part classification[C]//Robotics and Automation (ICRA), 2014 IEEE International Conference on. IEEE, 2014: 3143-3150. doi: 10.1109/ICRA.2014.6907311.

[15] Saupp é A, Mutlu B. Design patterns for exploring and prototyping human-robot interactions[C]. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2014: 1439-1448. doi: 10.1145/2556288.2557057.

[16] Prieto I, Izal M, Morato D, et al. Traffic generator using Perlin Noise[C]. Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE, 2012: 1847-1852. doi: 10.1109/GLOCOM.2012.6503384.

[17] Acosta M R G, Díaz J C V, Castro N S. An innovative image-processing model for rust detection using Perlin Noise to simulate oxide textures[J]. Corrosion Science, 2014, 88: 141-151. doi: 10.1016/j.corsci.2014.07.027.

[18] Dustler M, Bakic P, Petersson H, et al. Application of the fractal Perlin noise algorithm for the generation of simulated breast tissue[C]. SPIE Medical Imaging. International Society for Optics and Photonics, 2015: 94123E-94123E-9. doi: 10.1117/12.2081856.

[19] Nicholson C J, De Schreye D, Sneyers J. Improving compositions of the Apopcaleaps music generation system by using Perlin Noise[C]. Technical Communications of the 27th International Conference on Logic Programming, ICLP 2011,. 2011: 231-239. doi: 10.4230/LIPIcs. ICLP. 2011. 231.

[20] Marinescu A. Optimizations in perlin noise-generated procedural[J]. Studia Universitatis Babes-Bolyai, Informatica, 2012, 57(2). doi:10.1016/j.cageo.2015.02.010.

[21] Lacasta A, Juez C, Murillo J, et al. An efficient solution for hazardous geophysical flows simulation using GPUs[J]. Computers & Geosciences, 2015, 78: 63-72. doi: 10.1016/j.cageo.2015.02.010.