Multimed Tools Appl (2015) 74:259–270
DOI 10.1007/s11042-013-1758-4

Line recognition algorithm for 3D polygonal model using a parallel computing platform

Ji Hun Kang · Shin Jin Kang · SooKyun Kim

Published online: 12 November 2013

Abstract Line recognition-based rendering technique has been used effectively for shape transmission of 3D polygon model. Line recognition is defined by multifarious forms and characteristics of lines, and has been a fundamental key point in expressing shape of 3D polygon model in non-photorealistic rendering technique. Line recognition, however, requires a long period of calculation time and thus, various methods have been studied to accelerate the speed of the operation. This paper presents a new method that will accelerate the overall operation compared to the standard CPU-based method of extracting ink line. The new method will enhance the efficiency of the calculation speed by applying the parallel processing technique CUDA (Compute Unified Device Architecture) to the complex processes that consume a lot of time such as implicit surface calculation and feature point extraction. The overall performance will be tested and verified through various types of experiments with 3D polygon model.

Keywords Feature detection · Line recognition · 3D model · Parallel computing platform · GPGPU

1 Introduction

Characteristics of 3D polygon model are defined in multiple forms such as outline, silhouette, Ridge, and Valley. Among those characteristics, Ridge-Valley lines are considered to be one of the most significant and are studied in depth as they are used to identify appearance of the object. The problem with speed of the process, however, still remains as one of the factors that have to be improved. This paper will propose an approach of finding Ridge-Valley lines by using MLS surface approximation, as suggested by Kim et al. [4], and a feature extraction acceleration algorithm to shorten the overall computational time by using CUDA (Compute

J. H. Kang
Department of Computer Science Education, Korea University, Seoul, Korea

S. J. Kang
School of Games, Hongik University, Sejong, Korea

S. Kim (✉)
Department of Game Engineering, Paichai University, Daejeon, Korea
e-mail: kimsk@pcu.ac.kr

unified device architecture). Feature line extraction technique of 3D model discussed in the paper plays a fundamental role in non-photorealistic rendering technique. The technique generally has been used in ink line effect, which displays images as if they are manually-drawn by pen, as well as other different styles of rendering such as ink wash painting and oil painting. With an ink line effect of Ridge-Valley lines playing a fundamental role, distinct style of 3D polygon model are expressed in this paper through partial rendering or line shape modification.

CUDA-based line recognition acceleration presents a possibility of accelerating not only Ridge-Valley lines extraction but also other feature line extractions. In Section 2 of the following paper, we have explored over CUDA before we discuss the actual content of the study. In Section 3, we have illustrated how parallel processing of CUDA can be applied in feature extraction system, and especially propose CUDA-based MLS approximation and feature extraction system. We have explained the test result through table and figures in Section 4 and evaluate the result of the paper and discuss about the course of the future studies in Section 5.

2 Background

Feature line exists in multiple forms and definitions. Lines that are commonly used to represent feature line, such as Silhouette, outline, Ridge, and Valley, had been studied in various aspects for a long time. Feature line extraction is mainly divided in two separate categories. One of them is an object space-based approach method where feature line is extracted directly from the 3D model. The other one is image space-based approach method where 3D model is transformed into 2D image and feature line is extracted from the image. Among those, Ridge-Valley lines, the feature lines that were used in this research, have been studied in depth as well. Apparent Ridge proposed by Judd et al. [2] has been studied to demonstrate characteristic of a model and its related lines, and administers view-dependent feature line, which implies that feature line is extracted depending on how view changes. Hence, this approach can be inconvenient because line has to be extracted again every time the view changes. Another approach method that has been proposed is implicit surfaces fitting method. This method, proposed by Ohtake et al. [8], extracts view and scale-independent Ridge-Valley lines through first and second order curvature derivatives whose shapes are approximated by triangle meshes with high densities. Although this process is a visually excellent feature line extraction method, it does not produce good visual for the models with many wrinkles. It also requires the overall process to take around 1 h in order to extract Ridge and Valley from models that are composed of more than 40,000 triangles. The two studies that were previously discussed are object space-based approach. The following study is an image space-based feature line extraction. Line drawing method based on abstraction of a shaded image, as proposed by Lee et al. [5], is a GPU-based algorithm that renders lines along tone boundaries or thin dark areas in the shaded image of 3D model. This study achieves feature lines from 2D image and emphasizes artistic expression. As shown in the related works, feature line extraction has been studied in various aspects and is consistently advancing and improving.

3 Parallel computing technique using CUDA

This paper incorporates CUDA (Compute Unified Device Architecture), a type of GPGPU (General Purpose computing on Graphics Processing Units) technology developed by the hardware manufacturer company nVidia (https://developer.nvidia.com/category/zone/cuda-zone). In addition to the real number calculation ability that the original GPU was known for, CUDA uses thread-based parallel computing to accelerate the calculation speed, and has contributed in improving calculation rate [1].

CUDA's threads are structured in grid of many threads in one block. The number of thread block is determined by general size of processed data and number of system processor. Because all the threads within block must be on a common processor, they share limited amount of data. Hence, the number of threads in each block is restricted to 1,024. While the threads are processed, data can be accessed in various memory spaces. CUDA bases its calculation on data exchange between the main memory, called host, and GPU memory, called device and GPU is processed according to the instructions given by CPU. CUDA's calculation processes are operated in each individual thread and consequently have parallel structure where calculations are processed simultaneously [1]. By subdividing calculations of feature line extraction based on these structures, calculations that would be repetitive can be processed at the same time and therefore, reduce the overall computational time. Calculation can thoroughly be accelerated by using parallel computing.

Although CUDA's parallel processing has many benefits, there are also some limitations to it. When CUDA processes general calculations, it first inputs data from main memory to GPU memory and administers parallel processing through threads that are formed according to the instructions given by CPU. When the calculation is completed, CUDA concludes its calculation processes in cycle by transmitting the test results from GPU memory to the main memory [6, 7, 9]. Such data transmission structure can lead to collision during data delivery process due to system bus bandwidth and time delay [9]. There is also a possibility of an overflow of the data if a large quantity of data is copied all at once, and this can cause cancellation of memory copy. Algorithm process must also be prepared to be processed in parallel structure. If the parallel processing structure was not considered when preparing algorithm, each thread may compete each other when they access memory among each other, which can lead test result to be invalid. When the calculation is processed, there's also time consumed during data transmission because data must be transmitted between the main memory and GPU memory. Considering these structures of data transmission, calculations that require more works must be processed based on CUDA [9].

4 Line recognition using CUDA

4.1 Approximation surface and curvature calculation

This paper applies MLS (moving least square) surface approximation to extract feature lines. MLS creates a function using least-square error between a given vertex and an adjacent vertex. Figure 1 displays an overall flow chart of the algorithm proposed in the paper. Speed improvement algorithm proposed is divided in host and device section, which is appropriate for the characteristic of CUDA. The host inputs data, collects neighbor information, and transmits data to the device. In device, we calculate for curvature through MLS approximation and then find Ridge and Valley. Then finally, basic model and ink line are formed when result is transmitted from device to host.

After input data is transmitted from host to device, all calculations are processed through CUDA's kernel function, which is a function used in GPU. MLS surface approximation is formed by using method proposed by Kim et al. [4]. Surface approximation estimates approximate values through least-square error equation between vertex and it neighboring

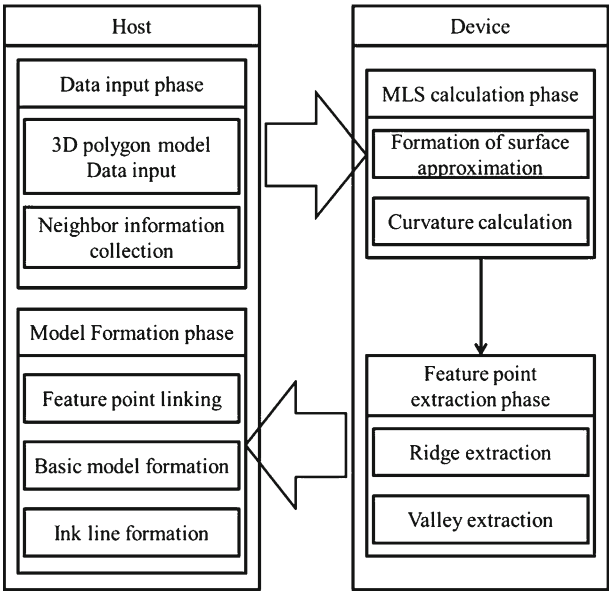


Fig. 1 Flow chart

vertex and applies those values to construct a hypothetical surface. Surface approximation formation is processed in accordance to all vertices and hence, threads are formed in accordance to the number of all vertices and surface approximation of all vertices is calculated through parallel processing. The following is an Eq. (1) used for surface approximation formation.

$$\sum_{i \in N_{r_j}} (p(x_i) - f_i)^2 \theta(\|r_i - r_j\|) \tag{1}$$

At MLS calculation phase, we form function that estimate least-square error between a vertex and its neighboring vertex. Then, surface approximation between all vertices and their neighboring vertices are formed and the curvatures and direction are calculated by using the Eq. (2) below.

$$e = \nabla k \cdot t = \frac{F_{ijl} t^i t^j t^l + 3k f_{ij} t^i n^j}{|\nabla F|} \tag{2}$$

The calculations that are repeated in all vertices are processed simultaneously using each individual thread. The main purpose of this calculation is to find surface approximation, which is defined by a set of approximate values between a specific vertex and its neighboring vertex. The proposed method studied by Kim et al. [4] is used in formation of this set. Figure 2 displays a structural diagram of a formation of CUDA-based MLS surface approximation.

As presented in Fig. 2, applying the fact that MLS surface approximation is formed from each vertex and its neighboring vertex, threads are formed in accordance to the number of all vertices and MLS surface approximation is formed through set of data that individually consists of information on each vertex and its neighboring vertex. Whereas standard method calculated surface approximation of all vertices through iterative processes, the new method can simultaneously calculate MLS surface approximation of all vertices through parallel processing.

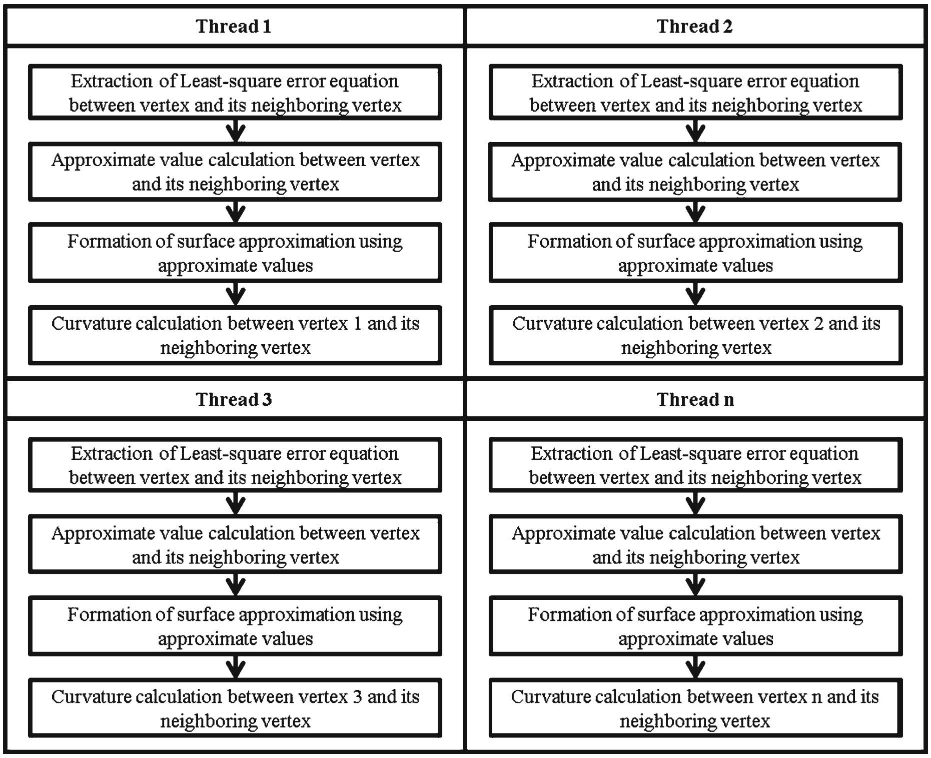

Thread 1
Extraction of Least-square error equation between vertex and its neighboring vertex
Approximate value calculation between vertex and its neighboring vertex
Formation of surface approximation using approximate values
Curvature calculation between vertex 1 and its neighboring vertex
Thread 2
Extraction of Least-square error equation between vertex and its neighboring vertex
Approximate value calculation between vertex and its neighboring vertex
Formation of surface approximation using approximate values
Curvature calculation between vertex 2 and its neighboring vertex
Thread 3
Extraction of Least-square error equation between vertex and its neighboring vertex
Approximate value calculation between vertex and its neighboring vertex
Formation of surface approximation using approximate values
Curvature calculation between vertex 3 and its neighboring vertex
Thread n
Extraction of Least-square error equation between vertex and its neighboring vertex
Approximate value calculation between vertex and its neighboring vertex
Formation of surface approximation using approximate values
Curvature calculation between vertex n and its neighboring vertex

Fig. 2 MLS surface approximation using CUDA

4.2 Detection of feature point

Ink line generally is formed by linking feature points, or vertex of Ridge-Valley lines. We use maximum curvature derivative in order to extract feature points [4].

Vertex of Ridge-Valley lines examine for all edges that exist inside 3D polygon model. Threads are formed in accordance to the number of all edges and each edge simultaneously processes feature point examination through parallel processing. When this examination is completed, each vertex can be distinguished whether or not they are Ridge-Valley vertex and feature point. Ultimately, the result of feature point extraction is transmitted from device to host. Calculation processes in device are completed when allocated memories in device are deactivated. Feature point extraction has similar structure as CUDA-based MLS surface approximation formation method. Threads are formed in accordance to the number of edges and each thread differentiates feature point through examination of all edges. The Fig. 3 presents a structural diagram of CUDA-based feature point extraction.

As shown in the above diagram, each thread is processed simultaneously, which can, therefore, reduce the overall computational time needed for feature point extraction.

4.3 Generation of line

Ink line is expressed in curve form through feature point linking. We have previously explained that calculations are processed in device and the test results are transmitted to host.

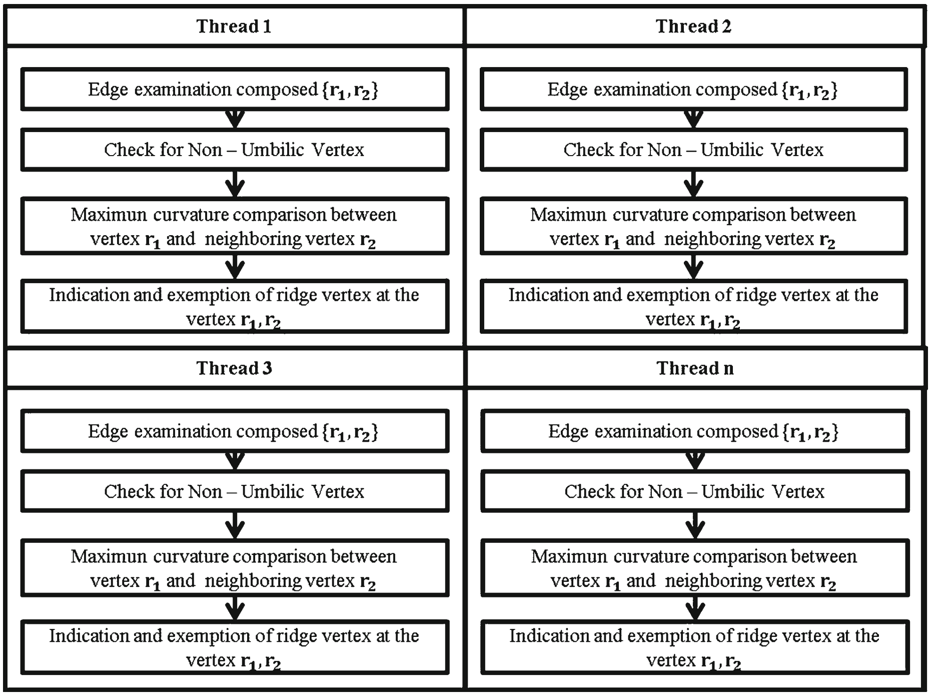

Thread 1
Edge examination composed $\{r_1, r_2\}$
Check for Non – Umbilic Vertex
Maximun curvature comparison between vertex r_1 and neighboring vertex r_2
Indication and exemption of ridge vertex at the vertex r_1, r_2
Thread 2
Edge examination composed $\{r_1, r_2\}$
Check for Non – Umbilic Vertex
Maximun curvature comparison between vertex r_1 and neighboring vertex r_2
Indication and exemption of ridge vertex at the vertex r_1, r_2
Thread 3
Edge examination composed $\{r_1, r_2\}$
Check for Non – Umbilic Vertex
Maximun curvature comparison between vertex r_1 and neighboring vertex r_2
Indication and exemption of ridge vertex at the vertex r_1, r_2
Thread n
Edge examination composed $\{r_1, r_2\}$
Check for Non – Umbilic Vertex
Maximun curvature comparison between vertex r_1 and neighboring vertex r_2
Indication and exemption of ridge vertex at the vertex r_1, r_2

Fig. 3 Detection of feature point using CUDA

The result values display information on feature point. After feature point information is transmitted from device to host, feature point linking is processed. In other words, because all of the CUDA-based calculations are completed in the earlier procedures, feature points are linked in host. Feature point linking randomly selects any feature point first and links it to other feature points that are not linked with neighboring vertices. If there are two feature points that are not linked to neighboring vertices, the one that will be selected for linking will depend on the direction. As these processes are repeated, feature line is connected. Also, Ridge and Valley lines link between each other. After the processes described earlier are completed, the two types of lines Ridge and Valley are linked. When line is linked, this could produce a short line, which is visually unappealing. Considering the length of ink line, these parts are eliminated from feature lines through critical values.

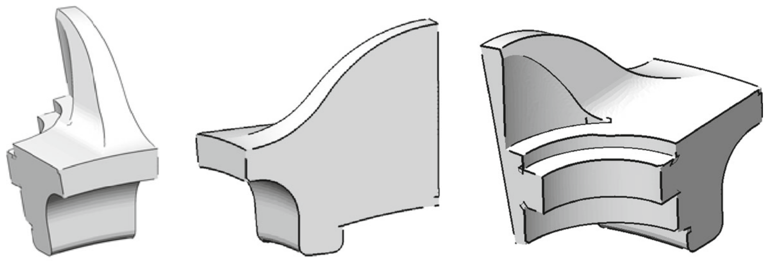
5 Experimental results

All the experiments tested in this paper were performed in Intel i7 3.2 Ghz CPU main memory and Window 7(x64) computer installed with Nvidia Geforce 285 GTX graphic card. For 3D graphic rendering, we have implemented C from OpenGL library and Visual Studio 2008, and have operated CUDA 4.0 SDK for parallel processing.

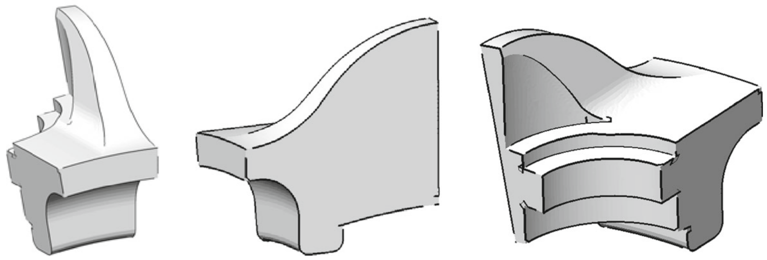
Table 1 compares the time difference between the standard method and the proposed method. The total column shows that the calculation speed was significantly accelerated in the proposed method compared to the old method. The bolded part displays the GPU-based calculation. Most of the calculations of feature extraction system are

Table 1 Result comparison table

Processing method	Model information			Computational time (unit: seconds)						
	Name of the model	Number of surface	Number of vertices	Neighbor info collection	Surface approximation formation	Ridge exploration	Valley exploration	Ridge linking	Valley linking	Total
GPU	Feline	32,000	16,053	0.1720	0.4357	0.0010	0.0012	0.0000	0.0000	1.4820
CPU				0.2500	6.7860	0.0150	0.0150	0.0160	0.0150	7.8920
GPU	Fan disk	12,946	6,475	0.0620	0.1558	0.0004	0.0005	0.0000	0.0000	0.7170
CPU				0.1090	2.6370	0.0010	0.0010	0.0160	0.0150	3.1050
GPU	Dino	31,999	15,998	0.1700	0.4233	0.0011	0.0013	0.0100	0.000	1.5000
CPU				0.2500	6.8300	0.0100	0.0100	0.0200	0.0100	7.9720
GPU	Bunny	31,999	17,195	0.1710	0.4124	0.0010	0.0012	0.0100	0.0160	1.5290
CPU				0.2660	6.8010	0.0160	0.0100	0.0160	0.0160	7.9210
GPU	Igea	32,000	16,002	0.1800	0.3930	0.0010	0.0013	0.0150	0.0160	1.5310
CPU				0.2960	6.6140	0.0160	0.0160	0.0150	0.0160	7.7840



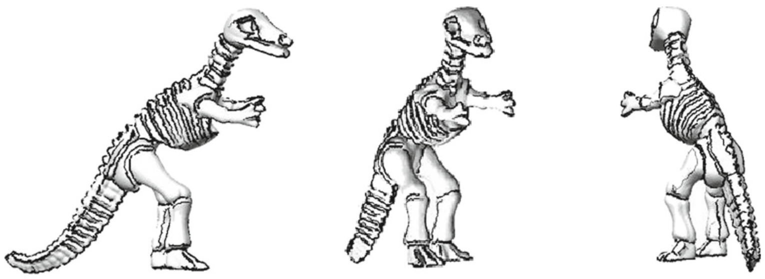
(a) Standard method (Computational time: 3.1050 seconds)



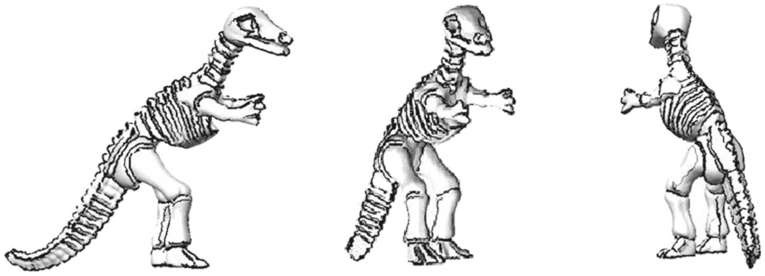
(b) Proposed method (Computational time: 0.7170 seconds)

Fig. 4 Fan disk (Number of faces: 12,964)

processed in GPU, which led to acceleration of overall calculation speed. Thus, even though there was a time delay due to data copy and transmission between memories

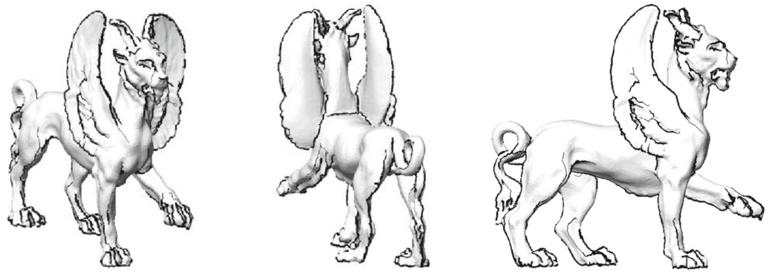


(a) Standard method (Computational time: 7.9720 seconds)

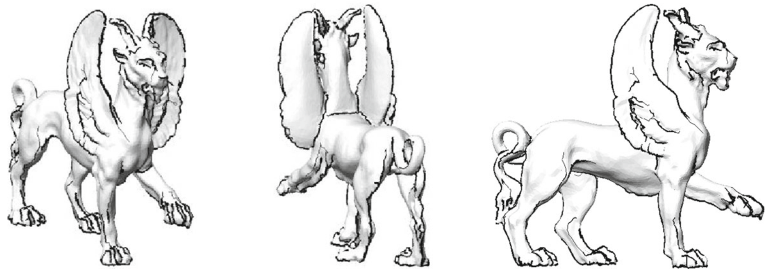


(b) Proposed method (Computational time: 1.5000 seconds)

Fig. 5 Dino (Number of faces: 31,999)



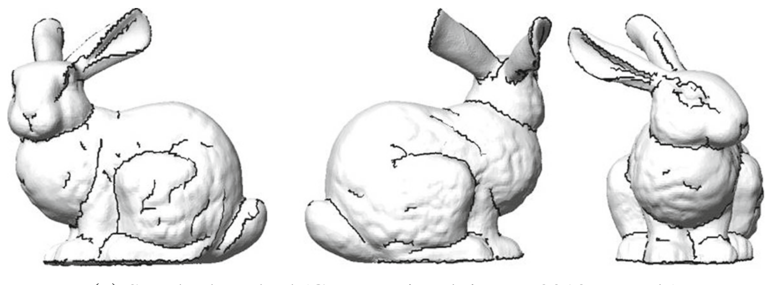
(a) Standard method (Computational time: 7.8920 seconds)



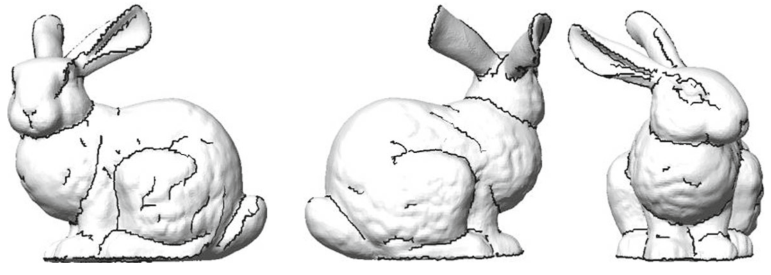
(b) Proposed method (Computational time: 1.4820 seconds)

Fig. 6 Feline (Number of faces: 32,000)

as previously explained, the total computational time was still exceptionally reduced. In computational Time, the sum of the time consumed from calculations is slightly

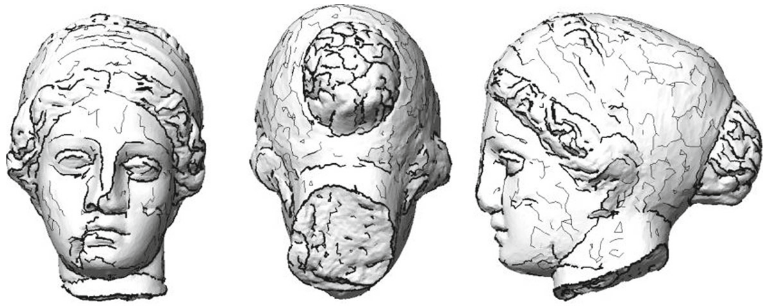


(a) Standard method (Computational time: 7.9210 seconds)

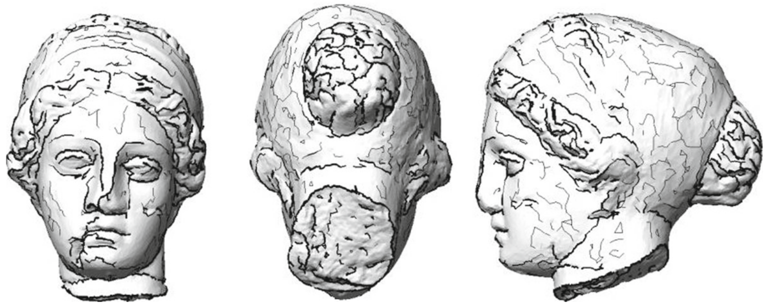


(b) Proposed method (Computational time: 1.5290 seconds)

Fig. 7 Bunny (Number of faces: 31,999)



(a) Standard method (Computational time: 7.7840 seconds)



(b) Proposed method (Computational time: 1.5310 seconds)

Fig. 8 Igea (Number of faces: 32,000)

different from the total because some time are consumed when data is input in main memory and when memories are copied and transmitted between host and device. Table 1 shows the for the model Feline, the total calculation took around 7.8 s to be completed when the standard method was used while the time dropped to around 1.8 s when the proposed method was used. MLS calculation, one of crucial factors in this rate comparison, took around 6.7 s when the standard method was used and was reduced to 0.4 s when the proposed method was used. For the model Fan disk, the total computational time was reduced from around 3.1 s to around 0.7 s and the duration of MLS calculation was reduced from around 2.6 s to around 0.5 s. For the model Dino, the total computational time was reduced from around 7.9 s to 1.5 s and the duration of MLS calculation was reduced from around 6.8 s to around 0.4 s. For the model Bunny, the total computational time was reduced from around 7.9 s to around 1.5 s and the duration of MLS calculation was reduced from around 6.8 s t around 0.4 s. The last model has also shown the similar result as well, with its computation time reduced from around 7.8 s to around 1.5 s and its MLS calculation time reduced from around 6.6 s to around 0.4 s.

Figures 4, 5, 6, and 7 display that the standard method and the proposed method both produced products with same quality. This shows that application of parallel processing helped to successfully reduce the overall computational time without any data simplification and omission (Fig. 8).

6 Conclusion and future work

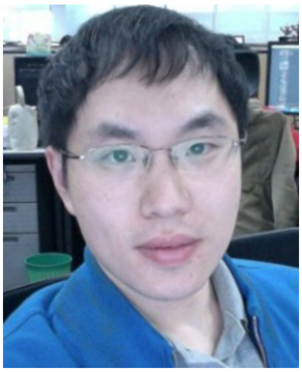
This paper proposed a method of accelerating calculation by applying CUDA to a line recognition algorithm that uses MLS surface approximation. The proposed method showed that we were able to significantly reduce the time consumed in feature extraction while maintaining the quality of the model. The speed improvement algorithm suggested in the paper is not only effective for Ridge-Valley line but could also be effective for acceleration of other form of feature lines such as silhouette and outline, and can possibly improve the overall speed of algorithm. In this paper, we have only applied ink line effect that displays the image as if it was manually drawn by a pen.

In the future, we plan to add other techniques such as feature line shape modification and texturing and apply effects other than ink line such as sketch effect and oil painting effect. Also we will apply our technique to 3D game algorithm [3].

References

1. Hwu W-M, Kirk D (2010) Programming massively parallel processors: a hands-on approach. Morgan Kaufmann
2. Judd T, Durand F, Adelson EH (2007) Apparent ridges for line drawing. ACM Trans Graph 26(3):19:1–7
3. Kang SJ, Kim SK, Cho SH (2011) Procedural game level generation with interactive evolutionary computation. J Future Game Technol 1(1):9–20
4. Kim SK, Kim C-H (2006) Finding ridges and valleys in a discrete surface using a modified MLS approximation. Comput Aided Des 38(2):173–180
5. Lee Y, Markosian L, Lee S, Hughes JF (2007) Line drawings via abstracted shading. ACM Trans Graph (ACM SIGGRAPH 2007) 26(3):18
6. nVidia (2011) NVIDIA CUDA C programming guide, nVidia
7. nVidia (2011) CUDA API reference manual, nVidia
8. Ohtake Y, Belyaev AG, Seidel H-P (2004) Ridge-valley lines on Meshes via implicit surface fitting. ACM Trans Graphics 23(3):609.12 [Proceedings of SIGGRAPH 2004]
9. Sanders J, Kandrot E (2010) CUDA by example. Addison-Wesle

Ji Hun Kang is a Ph.D. student in Department of Computer Science Education at Korea University, Korea. He received M.S in Department of Game Engineering at Paichai University, Korea, in 2013. His research interests include GPGPU, parallel processing, computer graphics and distribute data processing.



Shin Jin Kang received Ph.D. degree in Computer Science from Korea University in 2011. Since 2003, he has worked at Sony Computer Entertainment Korea and NCsoft as a lead game designer in various video games and MMORPGs including AION. He is now a professor at the School of Games at Hongik University. He is also the technical advisor of NCsoft.

SooKyun Kim received Ph.D. in Computer Science & Engineering Department of Korea University, Seoul, Korea, in 2006. He joined Telecommunication R&D center at Samsung Electronics Co., Ltd., from 2006 to 2008. He is now a professor at Department of Game Engineering at Paichai University, Korea. Professor Kim has published many research papers in international journals and conferences. Professor Kim has been served as Chairs, program committee or organizing committee chair for many international conferences and workshops; Chair of ICCCT'11, ITCS'10, HumanCom'10, EMC'10, ICA3PP'10, FutureTech'10, ACSA'09, Em-Com'09, CSA'09, CGMS'09, ISA'09, SIP'08, FGCN'08 and so on. Also Professor Kim is guest editor of the International Journal of "IET Image Processing" and "Multimedia Tools and Applications". His research interests include multimedia, pattern recognition, image processing, mobile graphics, geometric modeling, and interactive computer graphics. He is a member of ACM, IEEE, IEEE CS, KACE, KMMS, KKITS and KIIT.