Internal and External Analysis Considering the Layers of Three-dimensional Shapes Using CUDA

Satoshi Kodama[1], Yuka Ozeki[2], Rei Nakagawa[3]

[1]Research institute for Science and Technology, Tokyo University of Science, Japan

[2, 3]Department of Information Sciences, Faculty of Science and Technology, Tokyo University of Science

Abstract

With the development of three-dimensional (3D) printing, virtual reality (VR), and augmented reality (AR), a method to accurately determine the structures of 3D objects in various fields including computer-aided design (CAD) is required. However, unlike two-dimensional structures, analyzing 3D structures is highly problematic in terms of processing speed because the involvement of large number of data. In general, although it is possible to formulate an algorithm based on contact determination to achieve drawing at high speed, it is impossible with such an approach to capture the shape of the structure itself and therefore it cannot be used directly in CAD or 3D printing. Herein, to capture the structure, stereoscopic angles are used to perform internal and external determination for any point in three dimensions. However, the algorithm using the solid angle can be accurately internal and external determination, whereas a method using a conventional central processing unit with a simple triangular function is very problematic in terms of speed. From the above, to focus on the effectiveness of the parallel arithmetic when performing the internal and external determination in accordance with the algorithm using the stereoscopic angle, this study proposes a high-speed determination method using general-purpose graphics processing unit (GPGPU). In addition, by extending the idea of the 3D angle, it is shown that the shape can be captured accurately and quickly even for a complex shape (a 3D object including layering) inside another complex shape.

Keywords — *CUDA, Parallel Computing, GPGPU, Internal and External Analysis.*

I. INTRODUCTION

With the development of three-dimensional (3D) printing, virtual reality (VR), and augmented reality (AR), the need to capture accurately the structures of 3D objects including those with complex shapes also increases [1, 2]. In general, 3D analysis is difficult because it results in more data than do two dimensions, the main problem being the processing speed. To display objects at high speed in drawings or the like, contact-determination algorithms are often used for the 3D shapes [3, 4]. Although this method allows objects to be drawn at high speed by determining their shapes from the contact, the actual shape of a 3D object is not determined internally or externally at any point and therefore cannot be captured accurately. Therefore, the contact-determination method differs from general 3D computer graphics; to capture a 3D structure accurately, because it is necessary to calculate the shape including the internal structure, the amount of calculation is increased [5, 6].

II. RELATED RESEARCH

To determine any point of a 3D structure accurately both internally and externally, it is possible to extend the winding-number algorithm using the angle of the 2D version. Therefore, in this section, we explain the winding-number algorithm in two and three dimensions and the general-purpose graphics processing unit (GPGPU) used in this study.

A. An angle-based method for internal and external determination

1) Winding-number algorithm (two-dimensional)

Internal and external determination in two dimensions can be detected by the angle as a value with either positive or negative sign. Internal and external determination based on angle is shown in Fig. 1. It is possible to determine the internal point by setting the sum of the angles θ_1 to θ_7 from each vertex P_1 to P_7 to 2π (e.g., Fig. 2). Alternatively, it is possible to determine the external point by setting the sum of the angles θ_1 to θ_7 from each vertex P_1 to P_7 to zero (e.g., Fig. 3).

This method allows internal and external determination even for non-convex shapes. As in the above method, it is possible to determine whether the sum of the angle is 2π or zero (Figs. 4 and 5) [5-8].

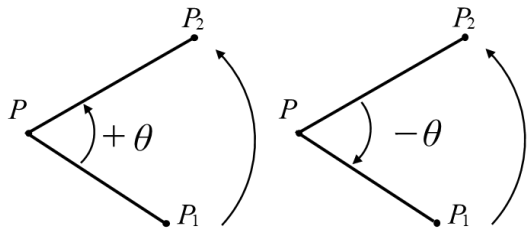

Fig 1. Positive angle and negative angle

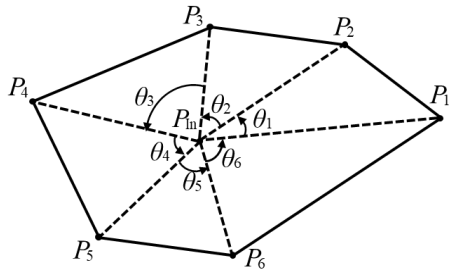

Fig 2. Example determined to be inside

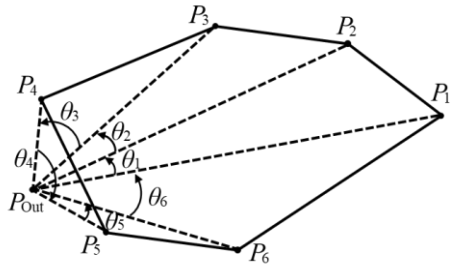

Fig 3. Example determined to be outside

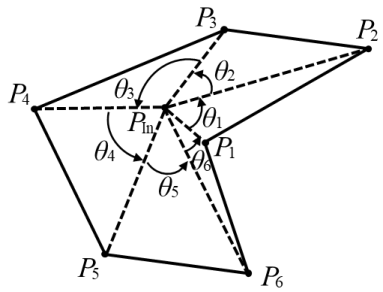

Fig 4. Example determined to be inside

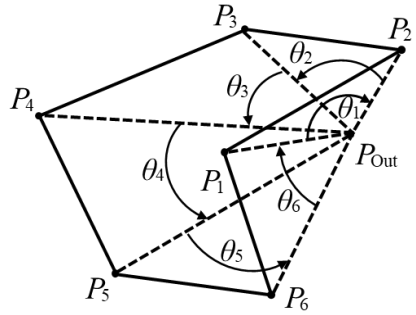

Fig 5. Example determined to be outside

From the above, we summarize that the internal and external determination of the number of vertices is *n*, in 2D shape, according to

$$w = \sum_{i=1}^{n} \theta_i = \begin{cases} 2\pi, & inner \\ 0, & outer \end{cases} \quad \cdots (1) \ .$$

*2) **Winding-number algorithm (three-dimensional)***

Using a solid angle for 3D objects allows for internal and external determination. When using a solid angle, with respect to shape A as shown in Fig. 6, using a sphere B of radius *r*, the entire sphere can be determined to be internal from being inclusive. By contrast, for shape A as shown in Fig. 7, when it is external, because it is not possible to project only a portion as a sphere C, it can be determined as external. From the above, it is possible to perform the determination. Meanwhile, for 3D space, we cannot define the order of data entry in a fixed direction, and entries must be in a fixed direction relative to each polygon. In other words, because positive and negative determination cannot be made as shown in Fig. 1, in the case of a solid angle (3D), determination is made as shown in Figs. 8 and 9 [5, 6, 8].

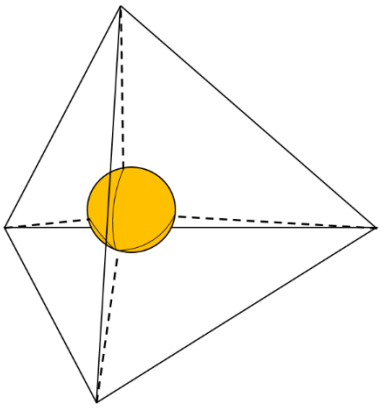

Fig 6. When inside

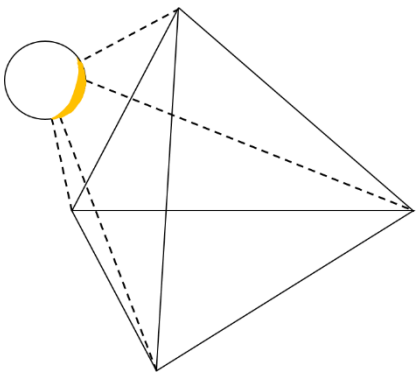

Fig 7. When outside

However, because this definition is in conjunction with the coordinate input method for polygons in general programming, it has high affinity with the algorithm in the present study [9-12].

For a specific calculation method, we use

$$S = \left(\sum_{i=1}^{n} \theta_i - (n-2)\pi \right) \times r^2 \quad (n \geq 3)$$

$$S_{all} = \sum_{j=1}^{m} S_i = \begin{cases} 4\pi, & inner \\ 0, & outer \end{cases} \quad (r = 1) \quad \cdots (2)$$

However, θ corresponds to Fig. 10 [6], the radius r is unity and m is the number for the polygon comprising the 3D object.

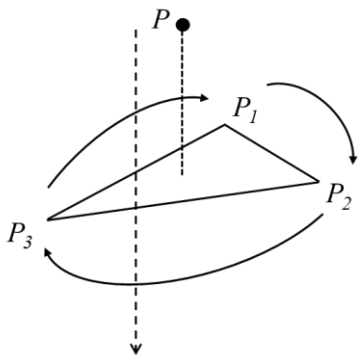

Fig 8. Example of positional determination negative in relation to a polygon

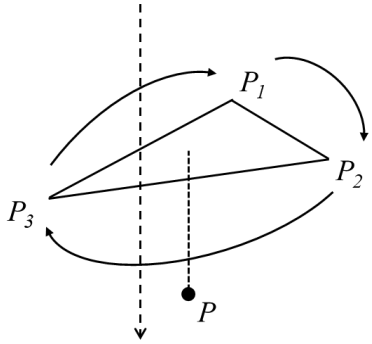

Fig 9. Example of positional determination positive in relation to a polygon

B. High-speed determination method using GPGPU

*1) **Development environment by CUDA using GPGPU made by NVIDIA***

CUDA is a parallel computation platform and a programming model for conducting general-purpose computation using a graphics processing unit (GPU) installed on a NVIDIA-made graphics board. However, CUDA often also refers to the programming model and language as well as NVIDIA's compiler and library, and in a wider sense it can indicate the set of software for running and executing the programming using an NVIDIA GPU [13-17].

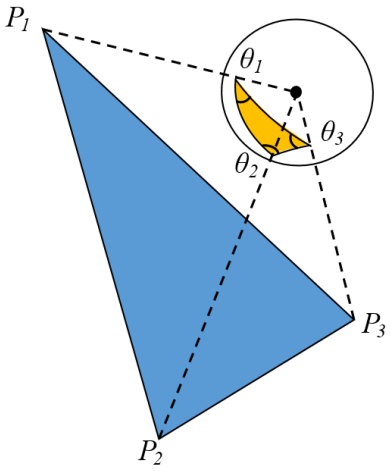

Fig 10. Projecting a polygon onto a sphere

By using a conventional central processing unit (CPU), because methods that use CUDA create huge numbers of threads, it is generally possible to obtain results faster. However, CUDA needs a separate device on a computer (Picture 1) [8], and we must write a program that has different hardware (device) awareness, as shown in Fig. 11 [13]. Two host codes running on the CPU side and a device code running on the GPU side must be created (Fig. 12) [13]. In addition, because the PCIe bus is used to connect to the device side to be processed by the host side and the GPU to be processed by the CPU, from the relationship of communication speed, it may slow down in some cases (Fig. 13) [13].

From the above, we understand, because it is not necessarily faster to use the GPU, that it is necessary to run a program that calculates the amount of algorithms and data.

Picture 1. PCI Express x16 Graphics Card (NVIDIA Quadro K420)

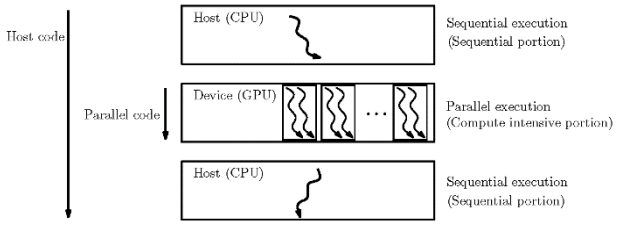

Fig 11. The CUDA programming model

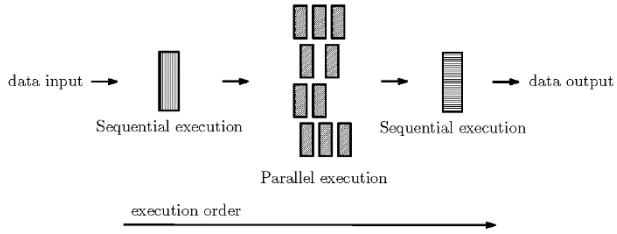

Fig 12. Sequential and parallel programming

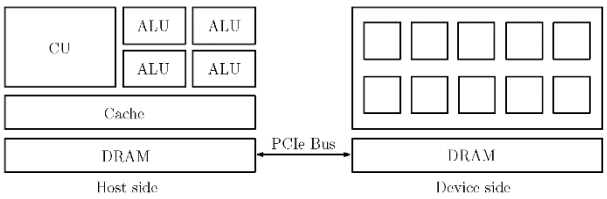

Fig 13. Heterogeneous Architecture

2) ***Parallel computation by GPGPU***

The structure of the GPU varies depending on the model and the core generation, but it basically comprises a large number of streaming multiprocessors in the GPU chip and the processing unit of the streaming processor that exists inside it. Although there are a number of processing units, it has a limitation that it is not possible to perform complex processing, unlike a CPU does. For example, control flow is a fundamental construct in any high-level programming language [13]. GPUs are comparatively simple devices without complex branch prediction mechanisms. Therefore, because the operating frequency of GPUs is generally low, they perform slower than CPUs.

CUDA is a single-program multiple-data programming model implemented on single-instruction multiple-data (SIMD) hardware [17, 18]. A large number of processors execute the same instructions in the thread program of the same kernel for each datum. In other words, unlike a multicore CPU, where different processors execute different instructions, in GPUs, all the processors execute the same instruction at the same time. Therefore, it is impossible to use GPUs for a general-purpose program, whereas independent, parallel operations with no dependencies between threads can be performed at high speed.

However, in the case of a general scientific and technical calculation, as the calculation involves dependency, the operation is performed efficiently by using SIMD and its extended version single-instruction multiple-thread (SIMT) [18, 19]. In the case of the SIMT architecture, however, the number of threads executed in a single instruction sequence differs, depending on the technology. For example, in the case of AMD, 64 threads are known collectively as a wavefront, and in the case of ARM, the Bifrost GPU is called the Quad, as collectively four threads [20-22]. Also, in the case of NVIDIA, which is used in the present experiments, 32 threads run collectively and are known as the warp. Therefore, it is necessary to develop systems in accordance with the use environment [19, 23-24].

3) ***Method for controlling threads using grids and blocks***

The CUDA, developed and provided by NVIDIA, performs at high speed by using a large number of threads. However, it is implemented using the grid size and thread block, because managing a large number of threads in a program is very difficult as well as not appropriate [13-16]. Therefore, CUDA exposes a thread hierarchy abstraction to enable one to organize one's threads. This is a two-level thread hierarchy that is further divided into blocks of threads and grids of blocks, as shown in Fig. 14 [13]. All threads spawned by a single kernel launch are referred to collectively as a grid, and a grid comprises many thread blocks. In the generation of threads, the grid is organized as a limit for the 3D array of blocks. Similarly, each block is organized as an upper bound for the 3D array of threads [13-16].

The number of threads that can be generated depends on the device's generation and model number, but if the device is the Pascal generation, then implementing it as a 3D sequence of the thread block (1024, 1024, 64) and the grid size (2^{31}–1, 65535, 65535) is possible [13-16, 23].

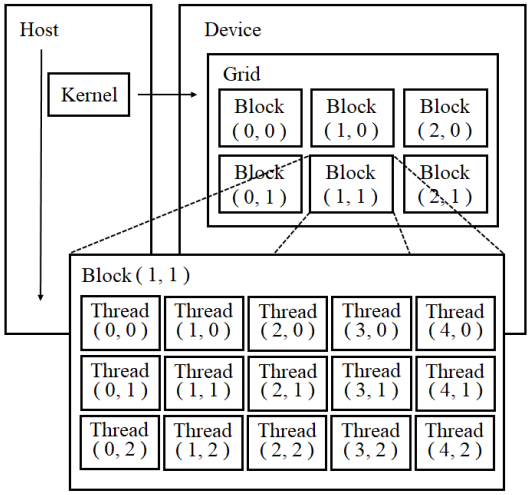

Fig 14. A two-level thread hierarchy

III. RESEARCH CONTENT

The purpose of this study is to make the internal and external determination fast and accurate for complex shapes with layering by using solid angles. With the algorithm using solid angles as described above, it is difficult to obtain results at high speed because the calculation is generally complicated. However, by using CUDA, it is possible to achieve the desired speed by creating many threads.

A. *Hierarchical determination of 2D objects based on winding-number algorithm*

By using the winding-number algorithm, it is possible to classify each closed surface by a hierarchy. According to the algorithm described above, if it can be determined as internal, then it becomes 2π. By extending this method, i.e., if there is a closed surface within the closed surface (i.e., a second layer), it can be seen that it becomes 4π because the apex is traced twice. Therefore, as shown in Fig. 15, area 1 is zero, area 2 is 2π, and area 3 is 4π, therefore it is possible to conduct internal and external determination including the hierarchy. The specific experimental results are shown in Fig. 16.

As explained previously, for a 2D complex shape having M points, the formula for internal and external determination considering the hierarchy is obtained by using the Eq. (3). However, k is the kth layer and θ_t, θ_{ij} is the angle as a value with either positive or negative sign. In addition, l is the number of the vertexes, the angle of vertex j of the object i is θ_{ij} and has either positive or negative sign:

$$w_{all} = \sum_{t=1}^{l} \theta_t = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_{ij} \right) = \begin{cases} 2\pi k, & inner(\ k\ th\ layer\ ) \\ 0, & the\ outer\ all\ objects \end{cases} \quad \cdots (3).$$

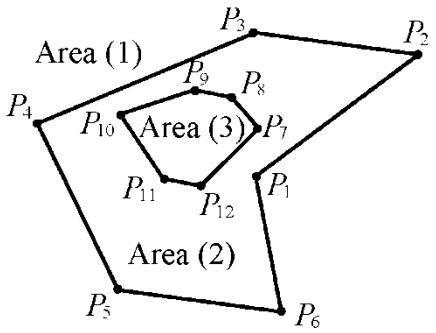

Fig 15. If necessary

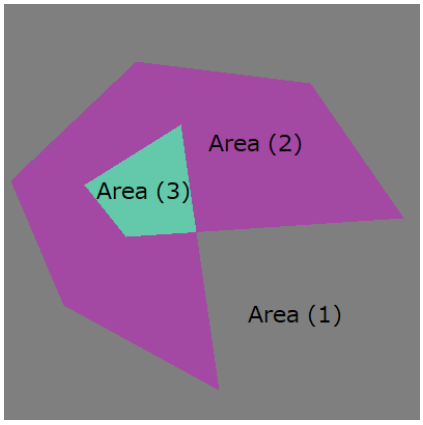

Fig 16. If necessary

$$w_{all} = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_{ij} \right) =$$

B. *Hierarchical determination of 3D objects based on winding-number algorithm*

By extending the 2D winding-number algorithm into three dimensions, it is possible to determine the hierarchy even in the case of using a solid angle. For example, as shown in Fig. 17, when a triangular pyramid (object 2) is contained inside a cuboid (object 1), it is 8π, as any point inside the triangular pyramid is inside two objects. By contrast, the inner point of only the cuboid (object 1), that is, not included in the triangular pyramid (object 2), is 4π. In addition, the point that is not included in any of the objects is zero. Therefore, it can be seen that area A is 8π, area B is 4π, and area C is zero. Similarly, in the case of Fig. 18, the part where objects 3 and 4 overlap, it is 8π. In addition, the point inside object 3 or 4 is 4π, and the point that is not included in any objects is zero. Therefore, in summary, area D is 8π, areas E and F are 4π, and area G is zero.

From the above, we deduce the following equation:

$$S_{kth(all)} = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} S_{ij} \right) = \begin{cases} 4\pi k, & inner(\ k\ th\ layer\ ) \\ 0, & the\ outer\ all\ objects \end{cases} \quad \cdots (4)$$

Here, k is the number of objects included, m is the total number of objects, and n is the total number of solid angles of each object.

IV. EXPERIMENTAL

As described above, by using GPUs, when processing the same calculation in parallel, it is possible to output the results at high speed even if the number of data is large. In the present study, we verified that the results can be output at high speed by using CUDA for a very large number of data calculations that use trigonometric functions such as solid angle by utilizing its properties.

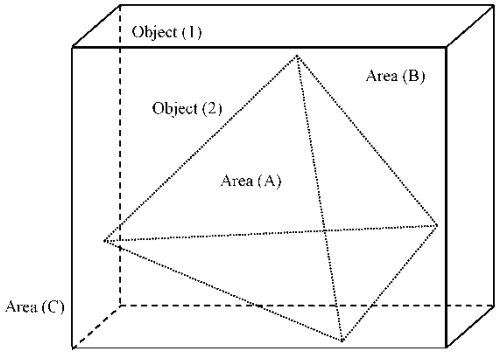

Fig 17. Relationship between objects and hierarchies (1)

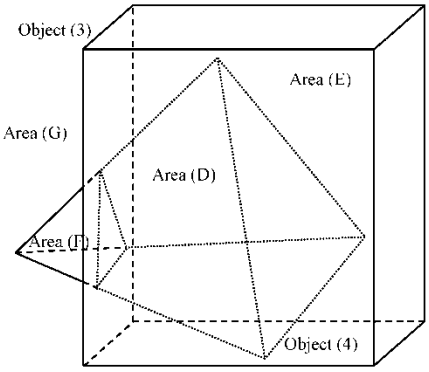

Fig 18. Relationship between objects and hierarchies (2)

A. Experimental content

The internal and external determination should be validated in the range $-50 \leq x \leq 50$, $-50 \leq y \leq 50$, and $-50 \leq z \leq 50$. However, the coordinates x, y, and z of the points to be determined are all integer values. Therefore, the number of points to be determined is $101 \times 101 \times 101 = 1{,}030{,}301$. In the experiments, internal and external determination was performed for the primitive 3D shapes, as shown in Figs. 19 - 24, the vertex and surface information of which is given in Table 1.

Because the proposed method is within the upper limit for thread creation, each decision point can correspond to one thread. Therefore, in implementing using CUDA, it was defined as a grid (16, 16, 16) and a block (8, 8, 8). However, in this case, because it exceeds the range determined from the relationship between the grid and the block [$(16 \times 8)^3 > 101^3$], we chose to discard any result outside the range.

From the above, the algorithm using CUDA was implemented according to the activity diagram shown in Fig. 25. In addition, for comparison, we implemented a method on a conventional CPU according to the activity diagram shown in Fig. 26.

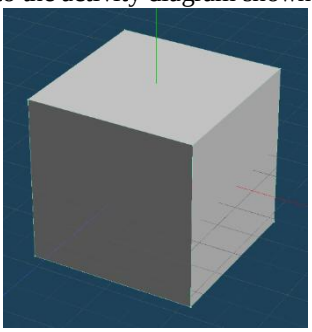

Fig 19. Box

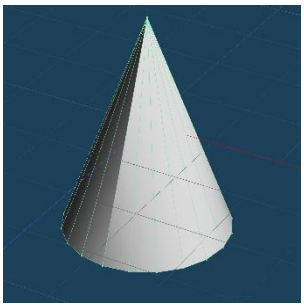

Fig 20. Cone

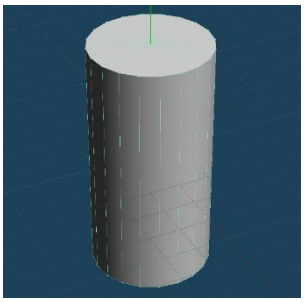

Fig 21. Cylinder

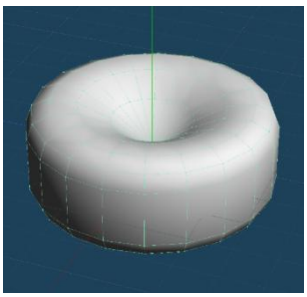

Fig 22. Convex shape

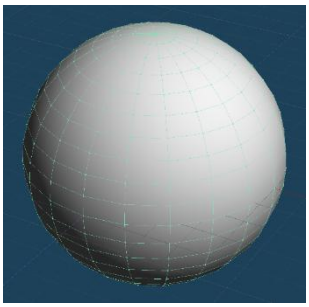

Fig 23. Sphere

Table 1. Information about primitive 3D shapes

	Vertices	Surfaces
Box (Fig. 19)	8	6
Cone (Fig. 20)	22	40
Cylinder (Fig.21)	42	60
Convex shape (Fig.22)	202	220
Sphere (Fig. 23)	382	400
Pipe (Fig. 24)	80	80

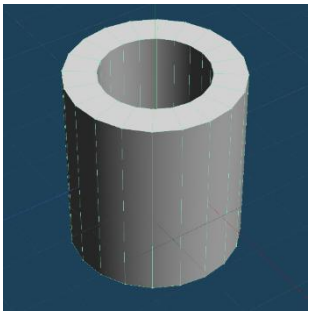

Fig 24. Pipe

B. Experimental environment

We used a GPGPU that can be incorporated into a laptop (NVIDIA 1060GTX). The experimental environment is a device that can be prepared relatively easily for consumers, and we reason that this environment is appropriate for verifying whether it can be applied to, in future, VR and AR fields. Specifically, we prepared the environment as described in Tables 2 and 3. More information about the device can be found in Fig. 27.

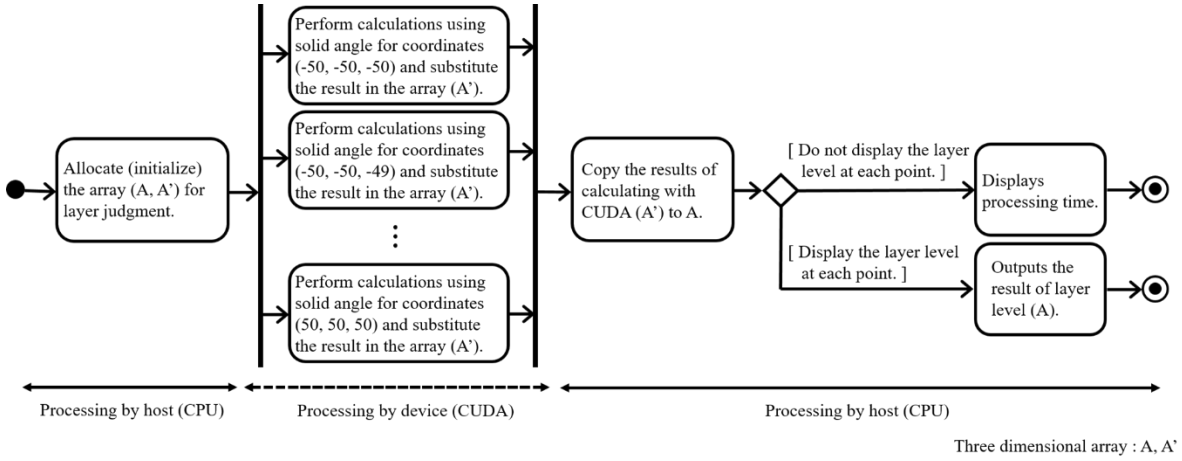

Fig 25. Activity Diagram

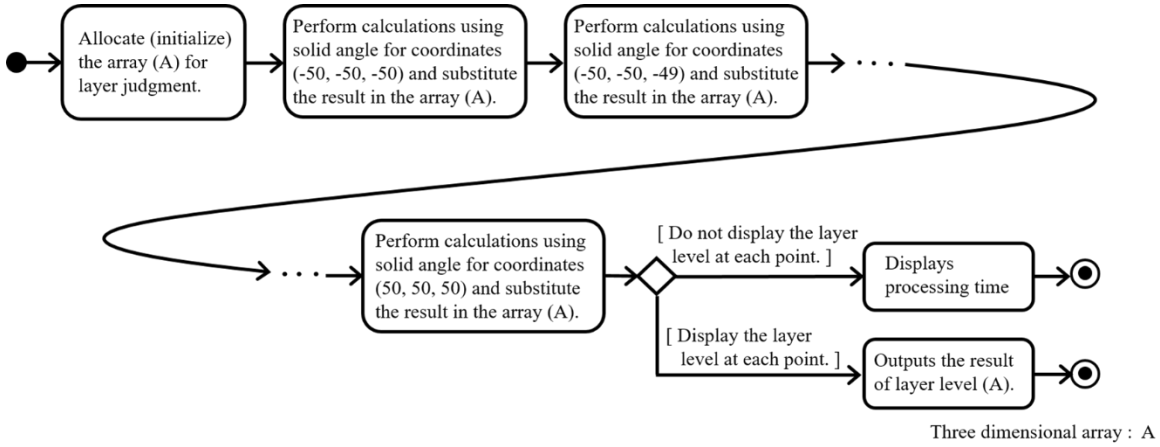

Fig 26. Activity Diagram

Table 2. Specifications of the host PC

Technical specification	Value
OS	CentOS Linux release 7.4
Compiler	gcc 4.8.5
CPU	Intel Corei7-7700HQ 2.80GHz
Memory	32 GB
SSD	512 GB

Table 3. Basic device information

Technical specification	Value
GPU	NVIDIA GeForce GTX 1060
Compiler	CUDA Toolkit 9.1.85

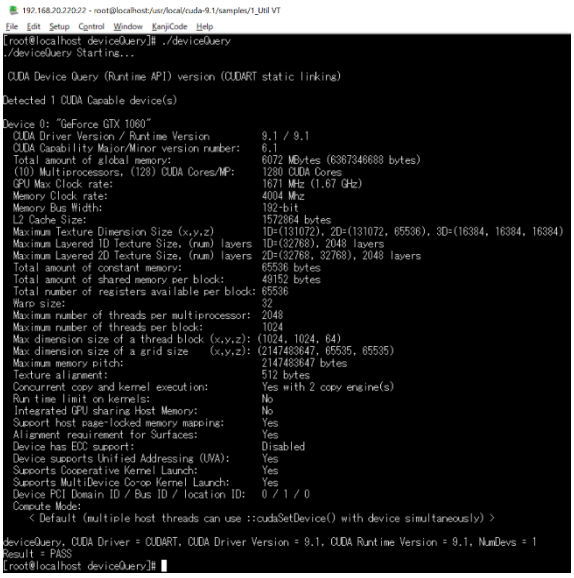

Fig 27. Specification of device used in this experiment

V. RESULTS

Table 4 gives the time taken to output the internal and external determination results for each 3D coordinate. In addition, the output results in consideration of the hierarchy in Table 3 are shown in Figs. 28–33. In outputting, the inside and outside determination result file of each coordinate is read and displayed using Java 3D. In all cases, the results could be outputted accurately at high speed.

Table 4. Specifications of the host PC

	Output results	GPGPU (s)	C (s)
Box(Fig.19) Cone (Fig. 20)	Figs. 28 and 29	0.991	24.198
Cylinder (Fig.21) Convex shape (Fig. 22)	Figs. 30 and 31	1.408	223.54
Sphere (Fig. 23) Pipe (Fig. 24)	Figs. 32 and 33	1.768	424.22

VI. CONCLUSION

To determine accurately the 3D structure of complex shapes, large amounts of data must be processed. In this study, we showed that using a GPU allows us to obtain the results at high speed. In general, because the numbers of vertices and surfaces to be the target of the operation to become complex shape increase, we reason that the proposed method is effective. However, because this method of using a GPU depends on the device, it is not necessarily the optimal method for a given shape. In future studies, we will seek the optimal thread generation system corresponding to the device so that decisions can be made at high speed for complex shapes in a wider range.

ACKNOWLEDGMENT

The authors would like to express our hearty thanks to the anonymous referees who pointed out several mistakes included in this original manuscript.

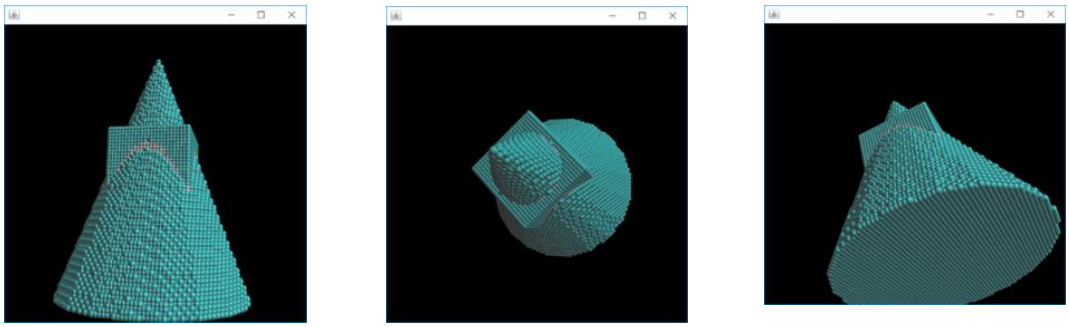

Fig 28. Sample data

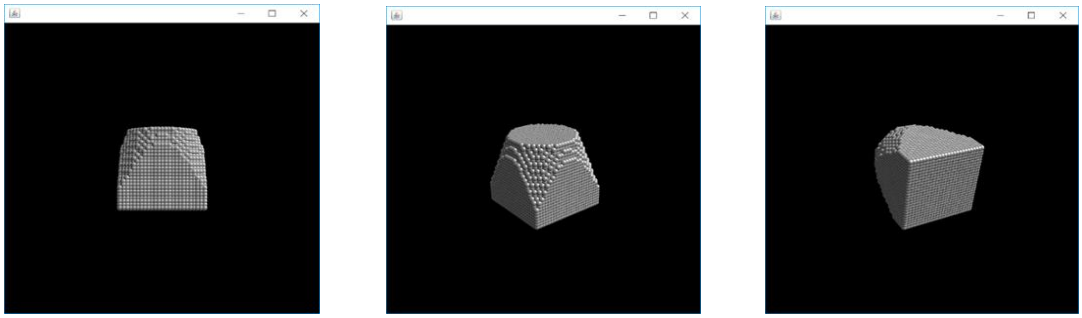

Fig 29. Sample data

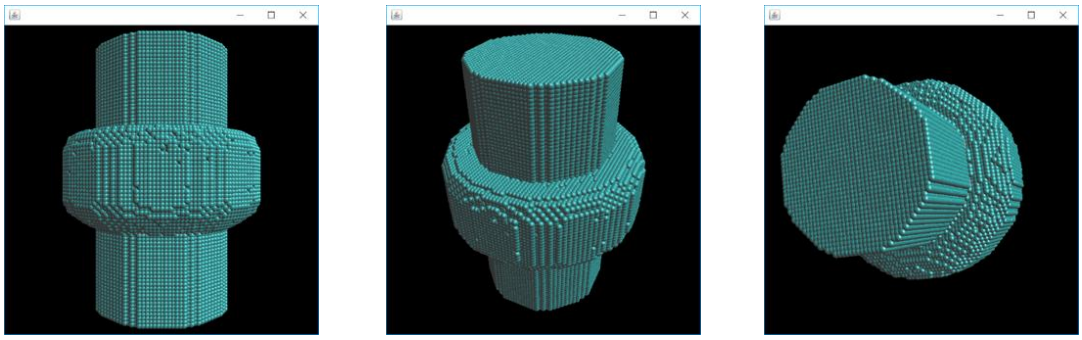

Fig 30. Sample data

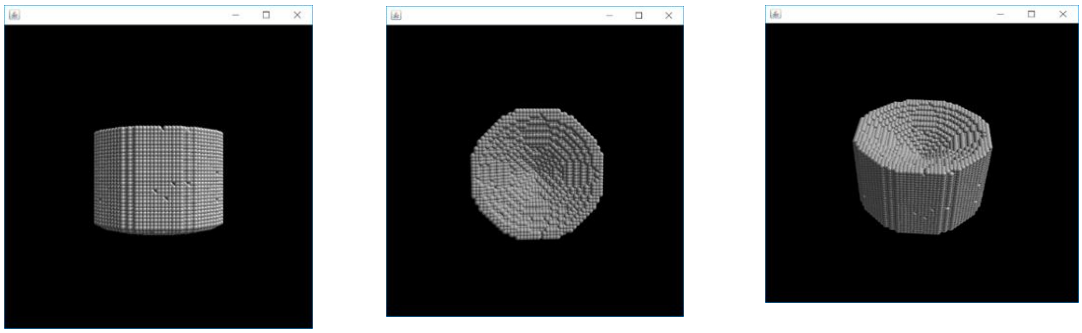

Fig 31. Sample data

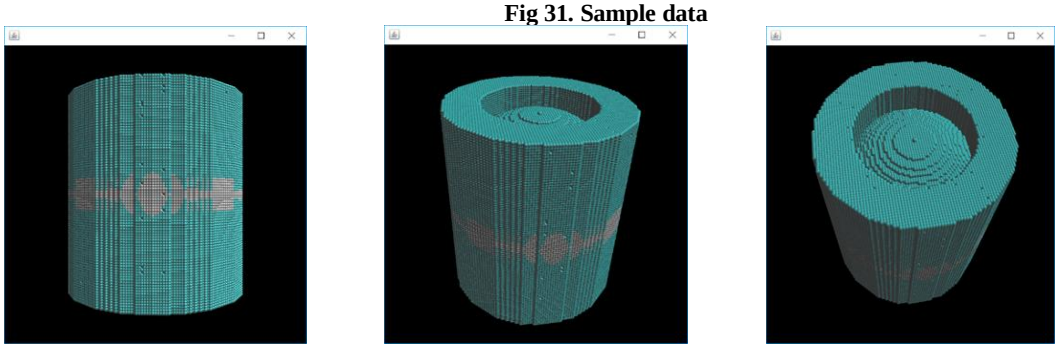

Fig 32. Sample data

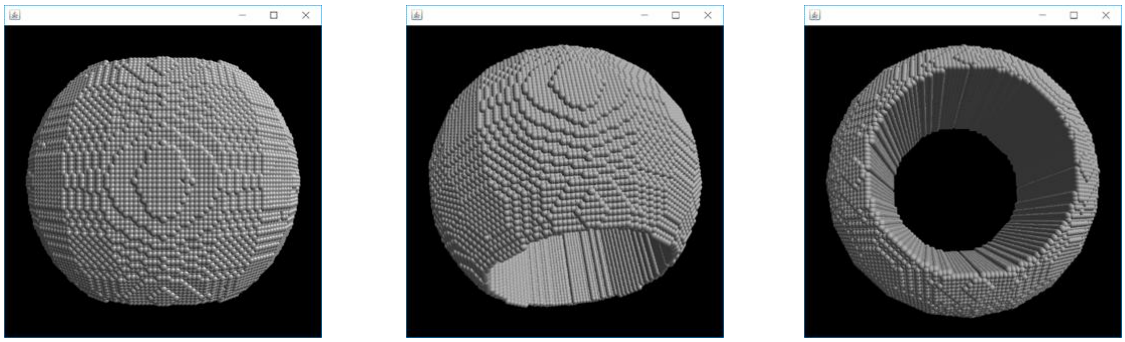

Fig 33. Sample data

REFERENCES

[1] THE NUNATAK GROUP, VIRTUAL & AUGMENTED REALITY, UPDATE, 4(7), 2016.

[2] Dusty Robbins, Chris Cholas, Mike Brennan, Keith Critchley, Augmented and Virtual Reality for Service Providers, Intel Corporation, Immersive media Business Brief, 2017.

[3] P. Jimenez, F. Thomas, C. Torras, 3D collision detection: a survey, Computers & Graphics, 25(2), 269-285, 2001.

[4] Y. Ozeki, S. Kameyama, S. Kodama, S. Akashi. A Proposal for the User Interface by Using Laser Devices Arranged in a Three Dimensional Space, Proceedings of IPSJ/IEICE Forum on Information Technology, Vol. 3, 385-388, 2016.

[5] Nakayama Atsushi, Kawakatsu Daisuke, Kobori Ken-ichi, Kutsuwa Toshirou, A Checking Method for a Point Inside a Polyhedron in Grasping an Object of VR, Proceedings of the 48th National Convention of IPSJ, 2, 297–298, 1994.

[6] S. Kodama, Verification of Efficacy of Inside-Outside Judgement in Respect of a 3D-Primitive Shapes Using GPGPU, International Journal of Modern Research in Engineering and Technology, 2(3), 1-11, 2017.

[7] Dan Sunday, Inclusion of a Point in a Polygon, http:// geomalgorithms.com/a03-_inclusion.html.

[8] S. Kodama, Effectiveness of inside/outside determination in relation to 3D non-convex shapes using CUDA, The Imaging Science Journal, DOI: 10.1080/13682199.2018. 1497251, 2018.

[9] Adrian Kaehler, Gary Bradski, Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library, O'Reilly Media, 978-1491937990, 2017.

[10] Samuel D. Jaffee, Laura Marie Leventhal, Jordan Ringenberg, G. Michael Poor, Interactive 3D Objects, Projections, and Touchscreens, Proceedings of the Technology, Mind, and Society, DOI: 10.1145/3183654. 3183669, 2018.

[11] Andrew Davison, Pro Java 6 3D Game Development: Java 3D, JOGL, JInput and JOAL APIs, Apress, 978-W1590598177, 2007.

[12] Richard G. Baldwin, Understanding Transforms in Java (Java Programming Notes # 1552), https://www. developer.com/java/other/article.php/3717101/Understanding-Transforms-in-Java-3D.htm, 2007.

[13] John Cheng, Max Grossman, Ty McKercher, Professional CUDA C Programming, Wrox Press Ltd., 9781118739327, 2014.

[14] David B. Kirk, Wen-mei W. Hwu, Programming Massively Parallel Processors, A Hands-on Approach 3rd Edition, Morgan Kaufmann, 9780128119860, 2016.

[15] Jason Sanders, Edward Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley Professional, 978-0131387683, 2010.

[16] Duane Storti, Mete Yurtoglu, CUDA for Engineers: An Introduction to High-Performance Parallel Computing, Addison-Wesley Professional, 978-0134177410, 2015.

[17] John Nickolls, GPU parallel computing architecture and CUDA programming model, IEEE Hot Chips 19 Symposium, DOI: 10.1109/HOTCHIPS.2007.7482491, 2007.

[18] Onur Mutlu, Computer Architecture: SIMD and GPUs (Part III) (and briefly VLIW, DAE, Systolic Arrays), Carnegie Mellon University, https://www.archive.ece.cmu.edu/~ece 740/f13/lib/exe/fetch.php?media=onur-740-fall13-module5. 1.3-simd-and-gpus-part3-vliw-dae-systolic.pdf.

[19] John Nickolls, William J. Dally, The GPU Computing Era, IEEE Micro, 30(2), 56 - 69, DOI: 10.1109/MM.2010.41, 2010.

[20] Jem Davies, The bifrost GPU architecture and the ARM Mali-G71 GPU, IEEE Hot Chips 28 Symposium (HCS), DOI: 10.1109/HOTCHIPS.2016.7936201, 2016.

[21] Avneesh Bhatnagar, Evan Speight, Dan Crawl, Joseph Dunn, John Bennett, Application management techniques for the Bifrost system, Proceedings of the 5th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2003), DOI: 10.1109/MCSA.2003.1240768, 66-76, 2003.

[22] S.J. Pennycook, G.R. Mudalige, S.D. Hammond, S.A. Jarvis, Parallelising Wavefront Applications on General-Purpose GPU Devices, Proceedings of the 26th UK Performance Engineering Workshop 2010, ISBN 9780955970320, 111-118, 2010.

[23] Daichi Mukunoki, Toshiyuki Imamura, Daisuke Takahashi, Automatic Thread-Block Size Adjustment for Memory-Bound BLAS Kernels on GPUs, IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC), DOI: 10.1109/MCSoC.2016.32, 2016.

[24] Zahid Ansari, Asif Afzal, Moomin Muhiuddeen, Sudarshan Nayak, Literature Survey for the Comparative Study of Various High Performance Computing Techniques, International Journal of Computer Trends and Technology, 27(2), 80-86, 2015.