Verification of Efficacy of Inside-Oute Judgement in Respect of a 3D-Primitive Shapes Using GPGPU

Satoshi Kodama¹

¹(Department of Information Sciences, Faculty of Science and Technology, Tokyo University of Science)

ABSTRACT: For the processing of data such as with 3D printing, Virtual Reality (VR) and Augmented Reality (AR), there is a need to seek technology which accurately and quickly analyzes the three-dimensional structures including that of complicated 3D forms. However, unlike in 2D situations when there are few data points, there is not yet an established method for processing it quickly for 3D forms due to the fact the objects constructing it are complicated as well as the fact that there is a lot of data points within the space. Generally, when illustrating a complicated form, a method is used whereby an object with the complicated form is generated using several primitive shapes. This method is used in various 3D modelling software because the position of the object can be intuitively and freely changed and since it can be easily written within DirectX or Java 3D, OpenGL, etc. In this thesis, it was shown that by using GPGPU (General-Purpose computing on Graphics Processing Units) in respect of an algorithm with a solid angle, the inside-outside judgement could be conducted quickly. Specifically, a measurement of inside-outside judgement processing was made for complicated shapes created from several primitive shapes as well as the measurement of processing time of several primitive shapes.

KEYWORDS – 3D printing, Inside-outside judgement, Parallel computation, Structural analysis, Virtual Reality

I. INTRODUCTION

Recently, in fields like 3D printing (3-dimensional printing), virtual reality (VR) or augmented reality (AR), there has been an increasing need to accurately and speedily process three-dimensional data including complicated shapes. However, unlike two-dimensional data, three-dimensional data has the difficulty in data notation such as there being lots of data or that it cannot be shown as one-dimensional rank, and hence there is no established method satisfying the above requirements.

Generally, for simple 3D drawing where focus is to draw quickly such as that used in games, etc., there are methods based on contact determination in relation to a particular area[1][2][3]. However, in structural analysis, etc. of objects or 3D printing, an accurate decision has to be made as to whether a particular point is within the shape or outside of it, but currently it is difficult to make that decision quickly.

II. RELATED RESEARCH

2.1 Inside-outside judgement algorithm in two dimensions

Inside-outside judgement relating to a point in respect of a given object has been researched a lot in the past. As particularly famous methods, there are methods called Crossing Number Algorithm[8] or Winding Number Algorithm[9] that are effective for two dimensional forms. Here the aforementioned method as algorithm relating to inside-outside judgement will be discussed in detail.

1. Crossing Number Algorithm

This is a method whereby inside-outside judgement is conducted by counting the number of times that a line drawn from the judgement point crosses the object in scope[4][5][6]. For example, as shown in fig. 1, in (a), there is an even number of crossing and in (b) there is an odd number hence they can be determined to be external and internal respectively. However, since this method can be processed very simply, there are cases when, although fast decision can be made, that incorrect judgement is made. For example, as shown in fig. 2, if the point to be judged is beyond the tip of the
shape (c) or on an edge (d), it’s not possible to conduct an accurate judgement.

2. Winding Number Algorithm [9]

This is a method for conducting inside-outside judgement using the angle between the start and end point of each edge and the point to be judged when tracing the edges of the target object centered around the point to be judged in one direction (either left-ward or right-ward)[4][6][7]. Specifically, it’s possible to determine whether the results obtained by using equation (1) is $2\pi$ or not.

$$w_n = \sum_{i=1}^{n-1} \theta_i$$  \hspace{1cm} (1)

However, $\theta_i$ is the angle between $P_0V_i$ and $P_0V_{i+1}$ (Fig.3). Moreover, Fig. 4 shows the case when it is deemed to be on the inside and Fig. 5 shows the case when it is deemed to be on the outside.

2.2 Expansion of algorithm for inside-outside judgement into three dimensions

The Crossing Number Algorithm and Winding Number Algorithm can be expanded to be three-dimensional. In this section, the inside-outside judgement method when conducting a three-dimensional expansion is explained in detail.

1. Crossing Number Algorithm (3D expanded version)

Generally, in the case of a three-dimensional object, inside-outside judgement can be made in respect of a particular point by processing each of the surfaces achieved after breaking it down into layers and processing via the two-dimensional version of the crossing number algorithm (Fig 6.). However, when using this method, in the end, a point P such as that shown in Fig. 7 which is on an edge needs to be found and the coordinate cannot be strictly obtained unless some geometric methods
are used, and therefore processing can take time.

Fig. 6: Case when processing is conducted by dividing a three dimensional form into a two dimensional plane

2. Winding Number Algorithm (Version expanded to 3D)

Inside-outside judgement can be made by applying a solid angle in respect of each vertex of a three-dimensional object[7][8][9]. When using a solid angle, it is possible to make decisions based on casting shadows from each vertex to a sphere. For example, if the point to be judged is as shown in fig. 8, since all of the sphere’s surface is covered hence it can be deemed to be inside.

A solid angle can be obtained using the following equation (2) in respect of a polygon with three vertices such as in fig. 9

\[ S = (\theta_1 + \theta_2 + \theta_3 - \pi) \times r^2 \] (2)

where \( r \) is the radius of a sphere.

Moreover, it's possible to expand the equation (2) above in respect of a polygon with \( n \) vertices as shown in equation (3) below and can be summarized as (4).

\[ S_n = \left( \sum_{i=1}^{n} \theta_i - (n - 2) \pi \right) \times r^2 \] (3)

\[ S_{all} = \sum_{j=1}^{m} S_j = \begin{cases} 4\pi, & \text{inside.} \\ 0, & \text{outside.} \end{cases} \] (4)

Here, in equation (4), \( m \) is taken to be the number of layers of polygons constructing the 3D object [14].

Fig. 7: Extracting a point crossing the plane

Fig. 8: Light shadow on object in 3D (if it can be deemed to be inside)

2.3 Existing methods for parallel computation

The detail will be discussed later but we propose a method for outputting results at a high
speed by conducting parallel computation by using CUDA (Compute Unified Device Architecture) [12].

CUDA is a parallel computation platform and programming model for conducting general purpose computation by using GPU (Graphics processing unit) installed on a NVIDIA made graphics board [14]. However, it often refers to ‘programming model’ and ‘programming language’ as well as their ‘compiler’ and ‘library’ and in a wider sense it can indicate the set of software for running and executing the programming using GPU of NVIDIA.

Generally, by using CUDA, parallel computation became possible to be conducted easily enabling results to be output quickly but on the other hand there is restrictions due to device and it is not necessary to be said that it’s effective for all algorithms [10][11].

The idea of a general purpose computation process by GPU is not a new concept, but it is a new concept if compared with existing data processing pipeline using CPU [11]. Therefore it needs to be done via a programming style using multi-core CPU, which is different from previous methods. In particular there are processes corresponding to several memory types (Tab. 1) and parallel block divisions (Fig. 10) for conducting parallel processing, and there are issues and problems that the developers face frequently.

1. The creation of thread and its processes

Parallel block division is important when programming using CUDA. CUDA differs a lot depending on the GPU architecture of different generations of device used but it’s possible to move a vast number of threads in whatever case. Therefore as it’s not sensible to manage multiple threads with one reference number, processing is done via a collection of threads and blocks as shown in fig. 10. For example, in the case of images and others it is easier to process them as a two dimensional matrix, and hence by programming in the form of \textbf{dim3 threads}(16,16) as a thread moving in a block with a block being \textbf{dim3 blocks}(2,1), a stepped structure such as that shown in Fig. 11 can be given allowing efficient generation.

Moreover, due to restrictions of the device, there are issues regarding the number of blocks which can be activated at once as well as an upper bound for the number of threads per block, and hence there is a need to appropriately create an algorithm corresponding to the purpose and hardware.

2. Characteristics of memory

There is a need to deal with the memory used suitably too. As a GPU is installed with many calculation units, it's more often that the bandwidth of the chip's memory becomes the bottleneck rather than the calculation throughput[12]. For this reason, there is a need to program bearing in mind the amount of memory traffic. In other words, it can be thought that by creating an algorithm which suits a particular situation enabling constant memory and others which are higher speed than a low-speed but high capacity global memory to be used well, results can be output more quickly.

| Tab. 1: An example of GPU memory used in CUDA programming |
|-------------|-------------|-------------|-------------|
| Capacity    | Global memory | Local memory | Texture memory | Constant memory |
| Speed       | Low speed    | Low speed   | High speed    | High speed     |
| Access from GPU | Read / write possible | Read / write possible | Read-in possible | Read-in possible |
| Access from CPU  | Read / write possible | Read / write not possible | Write-in possible | Write-in possible |
III. RESEARCH OVERVIEW

When using a three dimensional angle in general, since trigonometric functions are used a lot, there is a huge increase in the time spent on calculations. Hence in the past, processing had been done using a dedicated calculator [15]. However, as a result, it became unsuitable for application in devices, etc. which can be used easily such as that required as an extremely cumbersome experiment device becomes required.

Hence as the scope of research, on this occasion, we want to realize the speed-up of inside-outside judgement by using hardware that allows easy realization with versatility.

When using a modern and general CPU, by using a solid angle, the decision of whether a point is within a three dimensional object or not does not take too much time. However, on the other hand, calculating for all the points in space would take too much time. In other words, when conducting an inside-outside judgement of a particular point, there are independent calculation methods depending on the point hence it’s thought that it can be processed at a high speed by conducting parallel computations.

Hence in this research, the device for conducting parallel calculations is a small inserting type board that can be used with CUDA and a study was to be conducted of the efficacy of parallel computation using NVDIA’s Jetson TK1[13] (Fig. 12).

IV. EXPERIMENT

In order to show that processing can be conducted at high speeds by using parallel computation, the program for inside-outside judgement in respect of complex forms by using several primitive shapes was evaluated by actually creating it. The experiment environment and results are shown below.

1. Experiment environment

The main specifications of NVDIA’s Jetson TK1 which is the device used on this occasion mentioned in the previous section are shown in Tab. 2. Jetson TK1 DevKit uses Kepler architecture and is the first of the Tegra series to have Tegra K1 SoC (System-on-a-Chip) CUDA-corresponda[13][14]. Moreover, in terms of ability, the characteristic is that although it is inferior to a general GPU, it uses the same architecture as GPU for servers and workstations and desktops that have been announced by NVDIA [23]. Therefore Jetson TK1 is applied to insertion devices and is used in all sorts of situations such as insertion devices for car or image recognitions systems or devices for machine education, etc. [16]. Moreover, there is the advantage that the amount of electricity consumed is extremely small and it can be used easily and cheaply.

From the above, an environment was supposed in this research in which inside-outside judgement is required to be conducted quickly even for an insertion type device such as VR, AR or 3D printers, etc. and experiments were to be conducted using this Jetson TK1.
The basic development environment was created by installing Ubuntu 14.04 on this device and using CUDA6.5(GCC 4.8.4.). Moreover, the upper limit and others of CUDA in respect of this device were checked using deviceQuery(Fig. 13).

Moreover, an experiment was conducted in an environment such as that shown in Tab. 3 as hardware to be used in comparison with the existing speed.

2. Detail of experiment

On this occasion, objects were placed in spaces: \(-50 \leq x \leq 50, -50 \leq y \leq 50, -50 \leq z \leq 50\) as the target in the experiment and inside-outside judgement was to be conducted for the entire space \((1030301 (= 101 \times 101 \times 101)) \) points). Moreover, the primitive shapes placed were to be four triangular polygons making up a triangular pyramid (fig. 14), cone (fig. 15) or cylinder (fig. 16) and sphere (fig. 17).

![Fig. 13: Specification of device used in this experiment](image)

Tab. 2: Main specifications of JetsonTK1

<table>
<thead>
<tr>
<th>Scope</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>4 × A15 ARM core @2.3GHz (MAX) + NEON</td>
</tr>
<tr>
<td>GPU Core (Code name)</td>
<td>Kepler architecture (192core)</td>
</tr>
<tr>
<td>Memory</td>
<td>2GB DDR3 933MHz 64 bit CPU/GPU shared use</td>
</tr>
<tr>
<td>Interface, etc.</td>
<td>Gigabit Ethernet</td>
</tr>
<tr>
<td></td>
<td>HDMI1.4</td>
</tr>
<tr>
<td></td>
<td>SD/MMC card(a full-sizeslot)</td>
</tr>
<tr>
<td></td>
<td>GPIO</td>
</tr>
<tr>
<td></td>
<td>Half mini-PCIExpress</td>
</tr>
<tr>
<td></td>
<td>SATA</td>
</tr>
<tr>
<td></td>
<td>USB3.0</td>
</tr>
<tr>
<td>Electricity consumed</td>
<td>Less than 5watts</td>
</tr>
</tbody>
</table>

Tab. 3: Environment used for comparison

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel(R) Core(TM) i7-3960X CPU @ 3.30GH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>32GB (PC2-6400)</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 14.04.5 LTS</td>
</tr>
<tr>
<td>Compiler</td>
<td>GCC 4.8.4</td>
</tr>
</tbody>
</table>

![Fig. 14: Pyramid](image) ![Fig. 15: Cone](image) ![Fig. 16: Cylinder](image) ![Fig. 17: Sphere](image)
Specifically, the processing time was measured until the inside-outside judgement was made for each of the forms (fig. 14-fig. 17). Furthermore, by using several forms, the situation when complex forms are constructed was considered and the processing time in respect of a form such as below was also measured.

1. Union with each shape

1-1. Union of triangular pyramid (fig.14) and cylinder (fig.16)

1-2. Union of triangular pyramid (fig. 14) and sphere (fig. 17)

1-3. Union of cylinder (fig. 16.) and sphere (fig. 17)

Moreover, in order to check for situations such as when issues arise from overlapping objects, the processing times were measured for the following types of shapes (intersection) too.

2. Intersection with each shape

2-1. Intersection of pyramid (fig. 14) and cylinder (fig. 16)

2-2. Intersection of triangular pyramid (fig. 14) and sphere (fig. 17)

2-3. Intersection of cylinder (fig. 16.) and sphere (fig. 17)

Here, as aforementioned, the point which should be judged was to be processed as int type and each of the primitive shapes’ coordinate data processed as double type. A part of the coordinate data of Fig. 17 is shown in fig. 18.

3. Method of implementation

The programming was implemented based on Tab. 3 and Tab. 4 shown earlier. The basic algorithm for Fig. 14 - Fig. 17inaCUDAenvironment is shown as an activity diagram in Fig. 19. The grid size and thread size for implementing in CUDA were to be (16, 16, 16) and (8, 8, 8). Furthermore, the complex forms created as the collection of two primitive shapes were shown in Fig. 20 as an activity diagram. Furthermore, in this case too, two with grid size (16,16,16) shown earlier and thread size (8,8,8) were created and the running speed was to be improved by processing them in a non-synchronized manner. However, in either case, the calculated size for the whole lot will be 128 × 128 × 128 due to the grid and thread size hence the range outside of the size required on this occasion 101 × 101 × 101 was to be disposed.

As an existing algorithm made up of single thread used as comparison, Fig. 14-Fig. 17 have been shown in Fig. 21 as activity chart. Furthermore, complex forms created by a collection of two primitive shapes are shown as an activity diagram in Fig. 22. As a method of multithreadcreationthat’s beenused frombefore, the method using a general pthread was created with method based on Fig. 23 shown in an activity chart.

4. Experiment results

The mean times over five attempts of each experiment are shown in Tab.4–Tab. 6. However, when outputting the inside-outside judgement to a file, there is great dependence on the file access speed, etc. Hence when measuring the processing

Fig. 18: A part of the coordinate data of primitive shapes (fig. 17)

Fig. 19: Inside-outside judgement in respect of primitive shapes using CUDA
Capture (initialize) the array (A, B, C, A', B') for inside-outside judgement.

Use CUDA for the inside-outside judgement of one form. Substitute results in A'.

Use CUDA for the inside-outside judgement of second form. Substitute the results in B'.

Copy the calculated results (A') from CUDA to A.

Copy the calculated results from CUDA (B') to B.

[Union]
Output the union set from A and B to C

[Don't display processing time of inside-outside judgement]
Display the processing time

[Display results of inside-outside judgement]
Output inside-outside judgement result (A)

[Intersection]
Output the intersection set from A and B to C

[Don't display processing time of inside-outside judgement]
Display the processing time

[Display results of inside-outside judgement]
Output inside-outside judgement result (A)

[Display results of inside-outside judgement]
Output inside-outside judgement result (A)

Fig. 20: Inside-outside judgement in respect of two primitive shapes using CUDA

Capture (initialize) array (A) for inside-outside judgement.

Use CPU to conduct inside-outside judgement computation. Substitute result into A.

[Don't display processing time of inside-outside judgement]
Display the processing time

[Display results of inside-outside judgement]
Output inside-outside judgement result (A)

Fig. 21: Inside-outside judgement in respect of a primitive shapes using CPU

capture (initialize) the array (A) for inside-outside judgement.

Use CPU to conduct inside-outside judgement for the first shapes. Substitute the result into A.

Use CPU to conduct inside-outside judgement on the second shapes. Substitute the result of the union into A again.

[Union]
Use CPU to conduct inside-outside judgement on the second shapes. Substitute the result of the union into A again.

[Don't display processing time of inside-outside judgement]
Display the processing time

[Display results of inside-outside judgement]
Output inside-outside judgement result (A)

[Intersection]
Use CPU to conduct inside-outside judgement on the second shapes. Substitute the result of the intersect into A again.

[Don't display processing time of inside-outside judgement]
Display the processing time

[Display results of inside-outside judgement]
Output inside-outside judgement result (A)

[Display results of inside-outside judgement]
Output inside-outside judgement result (A)

Fig. 22: Inside-outside judgement in respect of two primitive shapes using CPU
Capture (initialize) the array (A, B, C) for inside-outside judgement

Use CPU to conduct inside-outside judgement of the first form. Substitute the result into A.

Use CPU to conduct inside-outside judgement of the second form. Substitute the result into B.

Output the union from A and B into C.

Display the processing time [Don’t display processing time of inside-outside judgement]

Output inside-outside result C

Display results of inside-outside judgement

Output the intersect from A and B into C.

Display the processing time [Don’t display processing time of inside-outside judgement]

Output inside-outside result C

Fig. 23: Inside-outside judgement in respect of two primitive shapes using pthread

Tab. 4: Processing time until the inside-outside judgement in respect of each form is complete

<table>
<thead>
<tr>
<th>Targeted forms</th>
<th>Output form</th>
<th>Number of vertices of the shape (Existing method (seconds))</th>
<th>Existing method (seconds)</th>
<th>Parallel computation (Proposed method) (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular (Fig. 14)</td>
<td>Fig. 24</td>
<td>4</td>
<td>2.76</td>
<td>5.22</td>
</tr>
<tr>
<td>Cone (Fig. 15.)</td>
<td>Fig. 25</td>
<td>40</td>
<td>24.05</td>
<td>35.47</td>
</tr>
<tr>
<td>Cylinder (Fig. 16)</td>
<td>Fig. 26</td>
<td>80</td>
<td>47.48</td>
<td>44.37</td>
</tr>
<tr>
<td>Sphere(Fig. 17)</td>
<td>Fig. 27</td>
<td>420</td>
<td>246.28</td>
<td>122.67</td>
</tr>
</tbody>
</table>

Tab. 5: Comparison in the processing time between the existing method using CPU and pthread and when calculating using CUDA (union)

<table>
<thead>
<tr>
<th>Targeted forms</th>
<th>Output form</th>
<th>Number of vertices on a form(Total)</th>
<th>Existing method (seconds)</th>
<th>Parallel computation (pthread)(second)</th>
<th>Parallel computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular pyramid + Cylinder</td>
<td>Fig. 28</td>
<td>84(4+80)</td>
<td>51.26</td>
<td>50.14</td>
<td>45.57</td>
</tr>
<tr>
<td>Triangular pyramid + Sphere</td>
<td>Fig. 29</td>
<td>424(4+420)</td>
<td>250.47</td>
<td>247.89</td>
<td>123.70</td>
</tr>
<tr>
<td>Cylinder + Sphere</td>
<td>Fig. 30</td>
<td>500(80+420)</td>
<td>299.26</td>
<td>296.64</td>
<td>140.71</td>
</tr>
</tbody>
</table>

Tab. 6: Comparison in the processing time between the existing method using CPU and pthread and when calculating using CUDA (intersection)

<table>
<thead>
<tr>
<th>Targeted forms</th>
<th>Output form</th>
<th>Number of vertices on a form(Total)</th>
<th>Existing method (seconds)</th>
<th>Parallel computation (pthread)(second)</th>
<th>Parallel computation (CUDA)(second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular pyramid + Sphere</td>
<td>Fig. 31</td>
<td>84(4+80)</td>
<td>51.32</td>
<td>50.43</td>
<td>45.60</td>
</tr>
<tr>
<td>Triangular pyramid + Sphere</td>
<td>Fig. 32</td>
<td>424(8+420)</td>
<td>250.46</td>
<td>297.87</td>
<td>123.73</td>
</tr>
<tr>
<td>Cylinder + Sphere</td>
<td>Fig. 33</td>
<td>500(80+420)</td>
<td>299.27</td>
<td>296.68</td>
<td>140.70</td>
</tr>
</tbody>
</table>
speed, only the processing time taken for the inside-outside judgement was actually measured based on the activity diagram (Fig. 19 – Fig. 23). Specific judgement result for Fig. 14 – Fig. 17 are shown in Fig. 24 – Fig. 27. Furthermore, 1-1 - 1-3 (forms shown via a union) shown in the detail of experiment in 5.2 are shown in order in Fig. 28 – Fig. 30. Moreover, 2-1 - 2-3 (form expressed as an intersection) are shown in order in Fig. 31 – Fig. 33. From the results, we see that the amount of calculation increases as the number of points increase hence its apparent that time takes time until the judgement is complete. Furthermore, the images in Fig. 24 – Fig. 33 are drawn by a program using Java3D separately created in order to visually grasp the coordinate data of the area inside output by this program.

V. EVALUATION AND TOPIC

If the number of vertices is greater than that in the experiment results, it became apparent that CUDA is better. From this, it’s thought that if the number of vertices are small, the CPU processing speed is generally high therefore although results can be obtained in a relatively short time but that if there are many vertices, parallel computation using CUDA in which thread can be generated for all points to be judged is more beneficial. Moreover, the conventional method using multithread (pthread) processes the inside-outside judgement of each form in parallel, so unlike the situation when inside-outside judgement is conducted in order for each shape, although high speed can be achieved, it’s slower than devices which can generate multiple threads such as a CUDA. In other words, with pthread which is conventionally used, if the processing is to be conducted for each form due to there being upper limit for a thread created such as in this case, it’s thought that parallel computation method using CUDA that can generate thread for each coordinate is faster.

From above, it’s thought that when conducting inside-outside judgement in respect of a large amount of three-dimensional data such as in this experiment, it is faster to conduct a parallel computation by generating multiple thread rather than conducting CPU processing. However, on the other hand, it is known that there is a big difference in speed arising from the programming model in respect of the branch or thread processing [10][11]. Moreover, since it is thought that appropriate data needs to be placed on the memory depending on the number of data, there is no guarantee that the program conducted in this experiment is the optimal solution. Hence, it’s thought that by constructing an optimal program that corresponds to the architecture, further increase in speed can be hoped for.

VI. Conclusion

Inside-outside judgement relating to a three-dimensional object is an important technology in respect of various fields such as 3D printer, VR or AR. However, three-dimensional data in general has a lot of data when compared with two-dimensional data hence existing methods cannot process them quickly.

On this occasion, solid angles were utilized as a method for conducting inside-outside judgement of three-dimensional data. This method uses triangle functions hence it can take a huge amount of time if conducting inside-outside judgement in respect of the entire of the imaginary space. In the past, speed was being increased by using devices such as expansion boards dedicated to scientific technique computations but on this occasion, we showed that it is possible to increase the speed by conducting parallel computation even if cheap devices used for drawing aimed at consumers are used. Moreover, it’s thought that in future, by improving the method, it would be possible to generate an object collecting several primitive shapes together and by further combining those objects, even more complicated forms can be easily represented.
REFERENCES


