A Study on Generation Method from Boundary Representation Model to Binary Voxel Model by Using GPU

Norihiro Nakamura
Osaka Institute of Technology
1-79-1 Kitayama,
573-0196 Hirakata, Osaka, Japan
nakamura@ggl.is.oit.ac.jp

Yusuke Inoue
3D Incorporated
UrbanSquea Yokohama Bldg.2F
1-1 Sakae-cho, Kanagawa-ku
221-0052 Yokohama, Kanagawa, Japan
inoue@ggl.is.oit.ac.jp

Yuji Teshima
Shizuoka Institute of Science and Technology
2200-2 Toyosawa
437-8555 Fukuroi, Shizuoka, Japan
teshima@cs.sist.ac.jp

Koji Nishio
Osaka Institute of Technology
1-79-1 Kitayama,
573-0196 Hirakata, Osaka, Japan
nishio@is.oit.ac.jp

Ken-ich Kobori
Osaka Institute of Technology
1-79-1 Kitayama,
573-0196 Hirakata, Osaka, Japan
kobori@is.oit.ac.jp

ABSTRACT

The boundary representation model and spatial partitioning model have been used in the computer graphics (CG) field. Each model has different advantages and disadvantages. Thus, if a user can mutually transform each model, the user can utilize the advantages of both. However, it is necessary to decide whether each voxel is inside or outside a shape when the user transforms the boundary representation model into the voxel model, and the processing load is large.

In this paper, we propose a fast method to transform the boundary representation model into the voxel model by using the Graphics Processing Unit (GPU); our method can decide the condition inside a shape. Furthermore, we propose an extended method that improves the accuracy of the voxel model generated.

Keywords
GPU, transformation shape, boundary representation model, voxel model

1. INTRODUCTION

When we represent a shape on a computer, we can classify the representation method as either the boundary representation model or the spatial partitioning model. A set of faces or curved surfaces is used to represent the shape in the boundary representation model. This model represents shapes with numerical formulas, which means it is accurate. In addition, the amount of data is relatively small when compared to the spatial partitioning model. However, it has some disadvantages. For example, users find it complex to use topological data for manipulating shapes, and the processing load for Boolean set operation, which users frequently use for modeling, is heavy.

On the other hand, there are several models based on the principles of the spatial partitioning model, one of which is the voxel model. This model has a simple data structure, and so the shape manipulation process is easy. Also, the Boolean set operation is easy. Therefore, a modeler based on this model has recently been proposed [Kas03]. However, when a user wants to improve the accuracy of representing shapes in this model, it is necessary to set a higher resolution. So, it needs a large quantity of data.

To solve the problems of each model, a data structure which involves the best features of both representations has been proposed [Yon96][Bru90]. However, the data structure of this approach is still complex. Another approach is to transform each model bi-directionally so that one can utilize the advantages of each model [Nak05]. This approach does have a problem: the processing load required to transform the boundary representation model into the voxel model is large. The reason is that it is necessary to decide whether each voxel is inside or outside the shape when the user transforms the boundary representation model into the voxel model. To solve this problem, some researchers have proposed methods which reduce the processing load required to decide the condition of each voxel [Kob95].

In regard to the processing involved in these representation methods, researchers have recently proposed faster methods by using the Graphics Processing Unit (GPU). The GPU is specialized hardware for drawing shapes, and it is used for fast processing or improvement of the accuracy of mathematical calculations [Ono03][Yam02][Yam04]. Also, we can alter part of the function of the GPU in accordance with our needs. Thus, a transformation method from a polygonal model to a voxel model by using the GPU has been proposed [Fan04][Hsi05][Don04]. Fan et al. proposed a method for transforming only the boundary of the shape into voxel data. However, it does not generate inside voxels and outside voxels. Hsieh et al. proposed the method for calculating a distance function with the GPU. Thus, the processing cost is large. The method proposed by Dong et al. is not robust in deciding whether voxels are inside or outside the shape.

In this paper, we propose a fast method to transform the boundary representation model into the voxel model by using the GPU; our method can distinguish the inside and outside of a shape. In our approach, the input data is the boundary representation model, which consists of triangle meshes. First, our method renders an input shape with an orthographic projection along the z-axis. Next, we get a particular color plane from the drawing result. Thus, we can generate a part of the voxel data. By repeating this process as many times as is the resolution of the voxel space along the z-axis, our method can generate the voxel data rapidly. In addition, we propose an optional method that improves the accuracy of the result.

2. RELATED WORK

Kobori et al. described how the spatial partitioning model is generated from the boundary representation model by using CPU only. This is a conventional method for transformation of the boundary representation model into the spatial partitioning model, as shown in Figure 1.

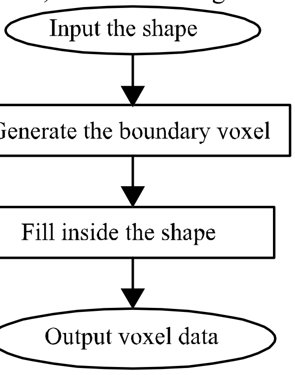


Figure 1. The flow of a conventional method

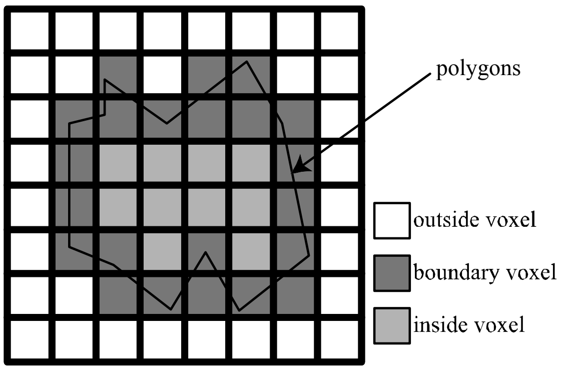


Figure 2. Boundary voxels

The steps for the conventional method are as follows.

Step 1: The method judges whether a voxel intersects the polygon that forms the boundary of a shape, and then it generates a boundary voxel, as shown in Figure 2.

Step 2: The method classifies each voxel into inside, outside or boundary voxels by filling the inside of the boundary voxels generated by Step 1.

Step 3: The method uses the inside voxels and the boundary voxels to represent the shape.

3. PROPOSED METHOD

3.1 Introduction

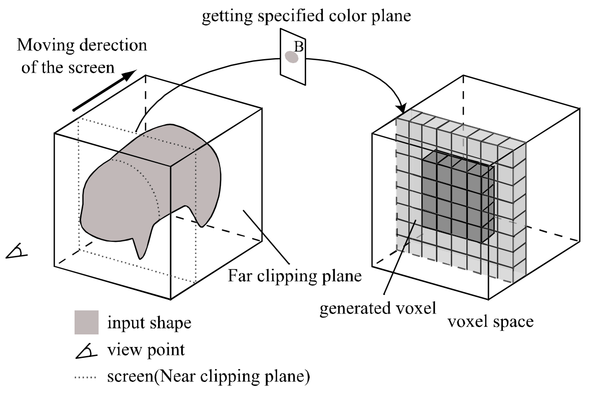

getting specified color plane
B
Moving derection of the screen
Far clipping plane
generated voxel
voxel space
input shape
view point
screen(Near clipping plane)

Figure 3. Proposed method

Our method draws a shape by orthographic projection after it sets the position of the screen, as shown in Figure 3. Furthermore, the initial clipping volume is a cube which circumscribes the bounding box of the input shape; in our method, the center of gravity of the bounding box is placed on the center of gravity of the initial clipping volume. Also, we define the screen as being constantly near the clipping plane of the clipping volume. Thus, our method draws the part of the shape that is placed behind the screen. If a pixel on the screen exists outside the shape, the color of its pixel is the color of the obverse face. If a pixel on the screen exists inside the shape, the color of its pixel is the color of the reverse face. Consequently, we can check whether the position of the pixel is inside or outside the shape by regarding the color of a pixel as the voxel. Therefore, it not only generates boundary voxels, but also generates the inside voxel and the outside one at the same time.

Our method repeats this process while moving the screen along the direction of the arrow in Figure 3. As a result, our method generates all of the voxel data. The processing speed of our method is faster than a method using only the CPU because the GPU has a specialized unit for the rendering process. Figure 4 shows the flow of the proposed method.

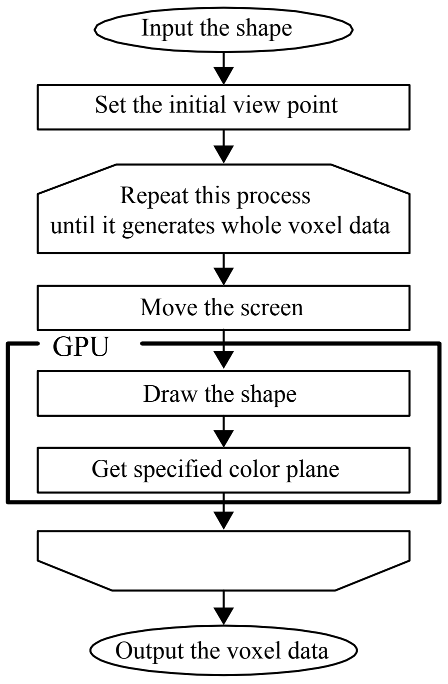


Figure 4. General flow of the method

Step 1: Set the resolution of the drawing space to the resolution of the voxel space.

Step 2: Set red and blue as the color of the obverse and reverse face of the input data, respectively, and set an initial view point. The color information is represented as (R,G,B) values: for example, red is (1.0,0.0,0.0).

Step 3: Draw the input data after moving the screen.

Step 4: Get the color information from the result of Step 3; our method gets only the blue color plane (B), as shown in Figure 3, and writes the result to the main memory. This result becomes the part of the voxel data corresponding to the position of the screen.

Step 5: Our method repeats Steps (3) and (4) until it generates all the voxel data.

3.2 Setting the position of the screen

Our method divides the draw space perpendicular to an arbitrary axis, as shown in Figure 5. The number of partitions is the same as the resolution of the voxel space. The position of the dashed line shows the screen. In addition, the distance between adjacent planes is the same as the length of a side of a voxel. For example, our method divides the draw space into 256 (slices) planes at a perpendicular angle to the z-axis in the case that the resolution of the voxel space is 256 x 256 x 256.

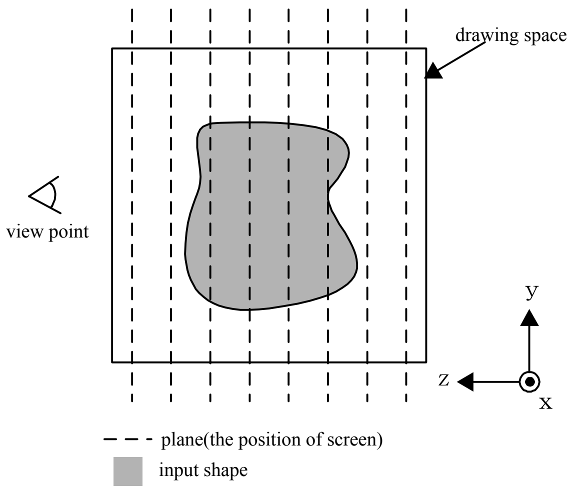


Figure 5. Screen position

Our method draws the shape by orthographic projection while moving the screen in the negative direction of the z-axis by the distance described above. At the same time, our method gets color information for the position of each plane.

3.3 Getting color information

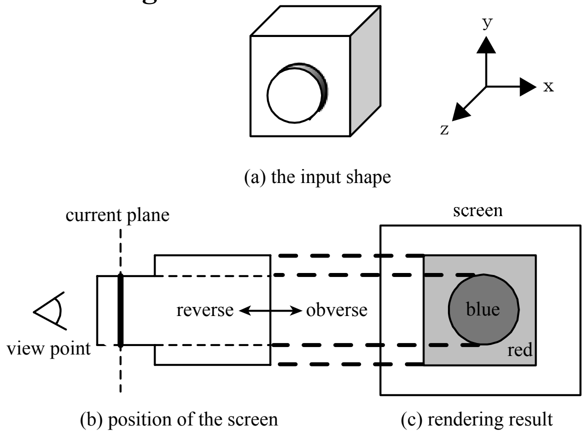


Figure 6. Relation between screen position and drawing result

Our method gets pixel data after drawing the shape at the plane described above. Figure 6 shows an example of the result when our method draws the shape at an arbitrary plane. The color of the reverse face is blue (0, 0, 1.0). Thus, when our method draws the shape in Figure 6(a), the color of the area corresponding to the heavy line in Figure 6(b) is blue (0, 0, 1.0), as shown in Figure 6(c). The position of a pixel colored (0,0,1.0) is inside the shape, and the position of a pixel colored (1.0,0,0) is outside the shape. Thus, getting the color plane of blue is the same as generating part of the voxel data. For example, we can assume that the blue color of the pixel data is part of the voxel data at Zi when we define the current z-position of the screen as Zi in Figure 7. Our method repeats this process for each plane.

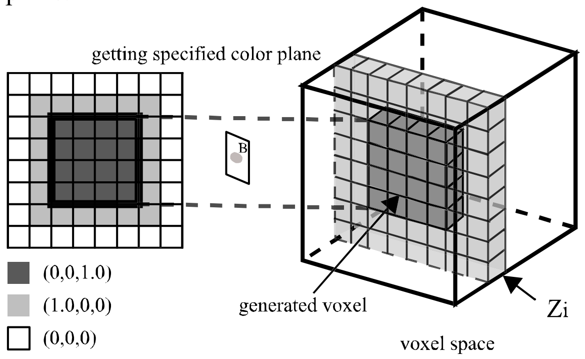


Figure 7. Generation of binary voxel data

3.4 Improving the accuracy of transformation

The GPU has the ability to alter its function in accordance with the developer's needs [Cg]. In this section, we propose a method that improves the accuracy of the transformation. When generating output pixel information to the screen, the GPU rasterizes an input shape based on the center of each pixel. Thus, our method sets a higher resolution for the image than the resolution of the voxel space, and the GPU rasterizes the shape. Our method increases the number of sampling points to improve the accuracy of the output voxel data.

In this paper, we set this resolution as twice the target resolution. This ratio (set resolution/target resolution) is defined as the magnification ratio. Our method uses the magnification ratio to divide a pixel. Then, we define the nearest plane from the view point as the first plane, and define the odd-numbered planes as odd planes and the even-numbered planes as even planes.

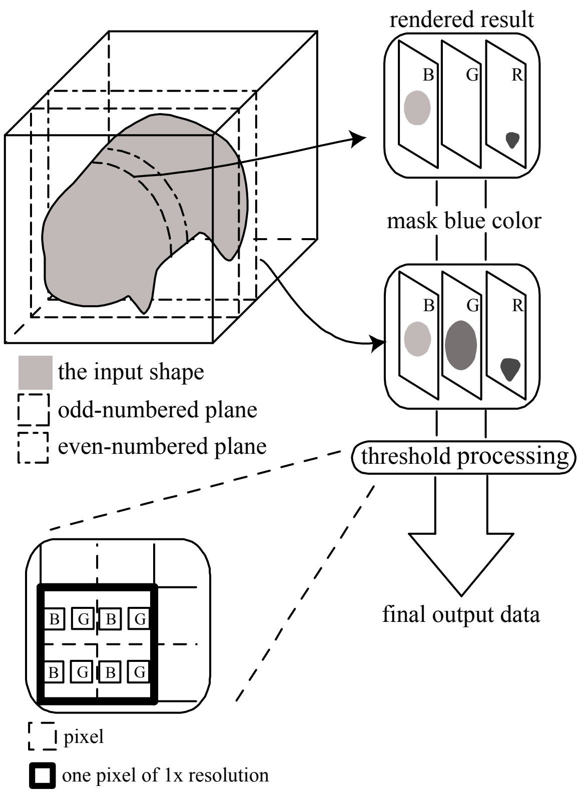
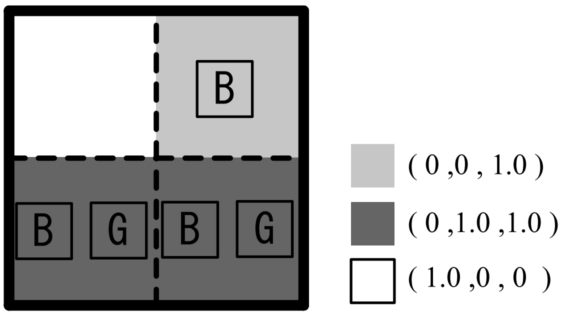


Figure 8. Improving the accuracy of the voxel data

First, our method rasterizes the input shape with the specified resolution; thus, the magnification ratio becomes 2x. This means we also increase the number of planes by two times the target resolution in accordance with the magnification ratio. The adjacent even planes and odd planes decide the condition of each voxel corresponding to a specified z-position. Our method renders the shape with two colors, red and blue, on an odd plane, as shown in Figure 8. Our method stores the rendered result to the GPU memory. Next, our method sets green (0, 1.0, 0) to the color of the reverse face, and masks the blue color. Then, it renders the shape on the even plane; in the even plane's rendering, our method combines the current result to the result of the odd plane's rendering. Consequently, the resulting data includes the color information of the odd plane's rendering.

Our method applies threshold processing to the rendered result of each even plane so that it generates the final output. We define the result of each even plane's rendering as the input buffer. Also, we define the buffer which stores the result of the threshold processing as the final output buffer. Our method makes these buffers in the GPU.

In threshold processing, it can be assumed that four pixels, surrounded by the heavy-lined frame in Figure 8, are one voxel because one pixel of the final output buffer comprises these four pixels. Our method sums the value of the blue and green of these four pixels; the total value is the same as the sum of the number of pixels that are inside the shape on each plane. The maximum value of the result is 8.0. Thus, it can be assumed that the pixel in the final output buffer corresponds to these four pixels and is inside the shape if the total exceeds the threshold. Repeating the threshold processing for each pixel in the final output buffer, our method decides the part of the voxel data corresponding to a particular z-position. Figure 9 shows an example of the threshold processing.



	▨	■	■	□		
B	1.0	1.0	1.0	0	=	3.0
G	0	1.0	1.0	0	=	2.0
					total	5.0

Figure 9. An example of the threshold processing

As shown in Figure 9, the total is 5.0. When setting 4.0 as the threshold, our method thus sets the pixel as blue (0,0,1.0). The color of one pixel in the final output buffer is decided by this processing. Finally, our method outputs the final output buffer to the main memory, as described in Section 3.3. By repeating the processes described in this section for as many times as the resolution of the voxel space along the z-axis, our method generates the voxel data to be transformed. These procedures, excluding the moving of the screen, are processed by the GPU. Figure 10 shows the flow for improving the accuracy.

In addition, our method can set 3x as the magnification ratio if we use the alpha information. In the case of more than three times magnification, it is possible to apply our method for the transformation by preparing a temporary buffer to store the result of the threshold processing.

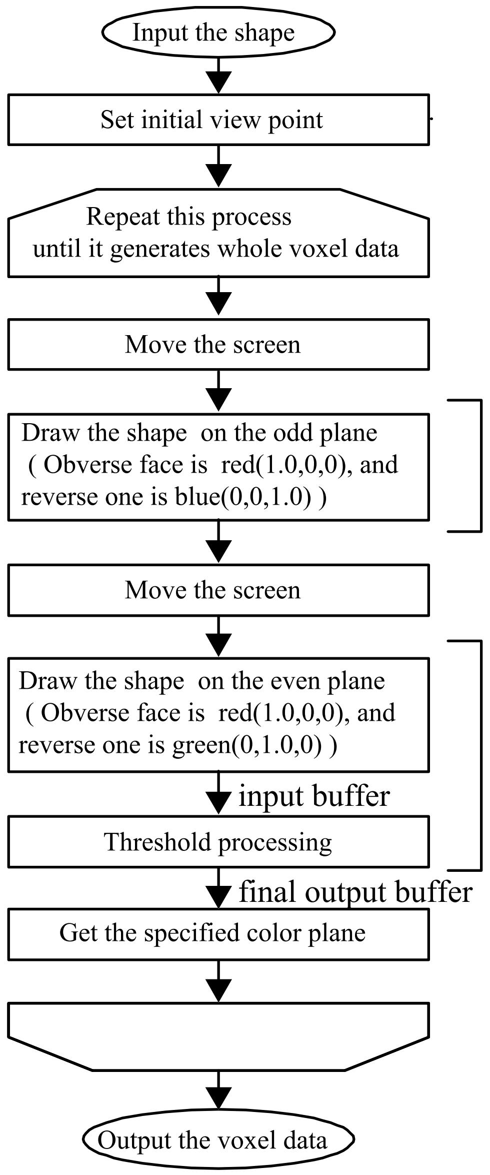
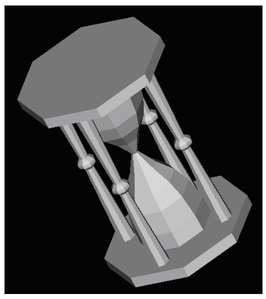

Input the shape
Set initial view point
Repeat this process until it generates whole voxel data
Move the screen
Draw the shape on the odd plane (Obverse face is red(1.0,0,0), and reverse one is blue(0,0,1.0))
GPU
Move the screen
Draw the shape on the even plane (Obverse face is red(1.0,0,0), and reverse one is green(0,1.0,0))
input buffer
Threshold processing
GPU
final output buffer
Get the specified color plane
Output the voxel data

Figure 10. The flow for improving the accuracy of the voxel data

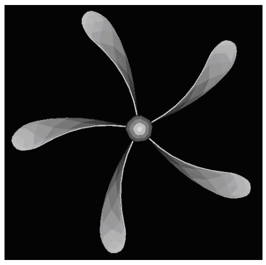
4. EXPERIMENT

4.1 Comparison of transformation speed

We experimented to verify the effectiveness of our approach. In the experiment, we measured the entire processing time of transformation. The measurement does not include the time for reading the shape data. We compared our method with the conventional approach described in Section 2 on a 3.2 GHz Pentium 4 computer and a GeForce6800 Ultra GPU. Figure 11 shows the experimental shapes.


(a) Hourglass (Face:1,136)

(b) Chair (Face:2,560)


(c) Propeller (Face:2,950)

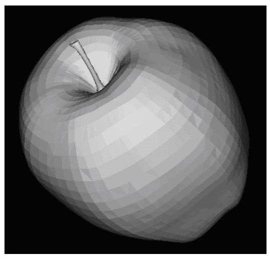
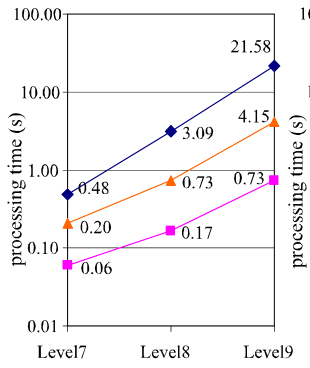
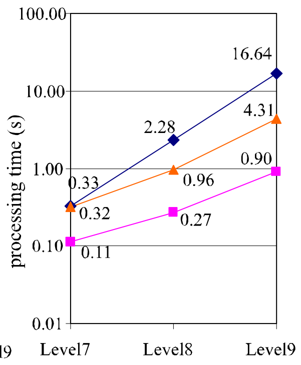

(d) Apple (Face:10,236)

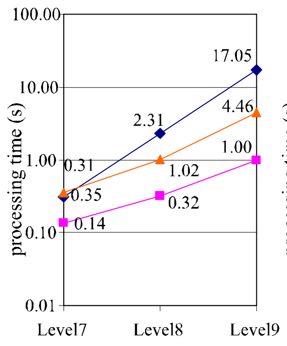
Figure 11. Experimental shapes



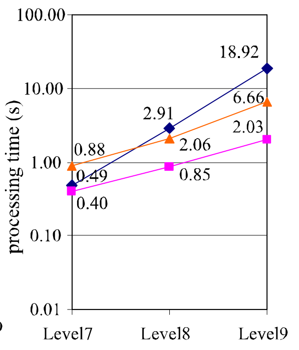
(a) Hourglass



(b) Chair



(c) Propeller



(d) Apple

◆ conventional method
■ without improving the accuracy
▲ including improving the accuracy

Figure 12. Relation of the generation time to the level

We set the level from 7 to 9; when we define the resolution of the voxel space as 2^n, n is the level. Also, we set 2x as the magnification ratio, and 4 as the threshold value for improving the accuracy.

Figure 12 shows the experimental results. The processing time in Figure 12 is shown by logarithmic graphs. Figure 12 shows that the effectiveness of our approach increases as the level increases. In the case of level 9, the processing time of our method is only 20-35% of the processing time of the conventional method when we apply the process of improving the accuracy to the transformation process. Also, our method's processing time, excluding that for improving the accuracy, is only 18-31% of that when applying the method to improve the accuracy. Also, our method becomes faster than the conventional method as the number of triangles increases.

Table 1 shows a breakdown of the processing time when applying our method to the apple shape; this table includes the processing time for improving the accuracy. Table 1 shows that the processing time of drawing the shape accounts for over 50% of the entire processing time for transforming the shape into voxel data at level 9.

	Get the color plane	Threshold processing	Rendering shape
Level 7	4.71	4.83	90.46
Level 8	12.31	10.06	77.63
Level 9	28.40	21.21	50.39

(%)

Table 1. Classification of the process for the apple shape, including the time for improving the accuracy

4.2 Validation of the accuracy

In this section, we verify the effectiveness of the method to improve the accuracy. First, we applied our method to the four shapes shown in Figure 11 without improving the accuracy. We compared the error value of the result with the error value of the voxel data to which we applied our whole method. The error value is the shortest distance between the boundary of the shape and the gravity point of each voxel corresponding to the boundary of the result. We obtain the maximum error and the average error of the distance by Eq.(1).

$$MaxError = \max_{voxel \in V} (\text{MinDist}(voxel))$$

$$AveError = \frac{1}{N} \sum_{voxel \in V} (\text{MinDist}(voxel))$$

$$\text{MinDist}(voxel) = \min_{face \in F} (\text{Dist}(face, voxel)) \qquad \ldots(1)$$

V : the set of voxels corresponding to the boundary of the result

F : the set of triangles of the input shape

N : the number of members of the set V

where Dist(*face*,*voxel*) is the function for obtaining the shortest distance between the *face* and *voxel*; *voxel* is the gravity point of a voxel: MinDist(*voxel*) is the function for obtaining the minimum distance in the set of the shortest distances between each triangle and *voxel*. The error is based on the length of a voxel. The level for the four shapes shown in Figure 11 was set at 8. The magnification ratio and the threshold value were the same as the settings described in Section 4.2. Figure 13 shows these experimental results.

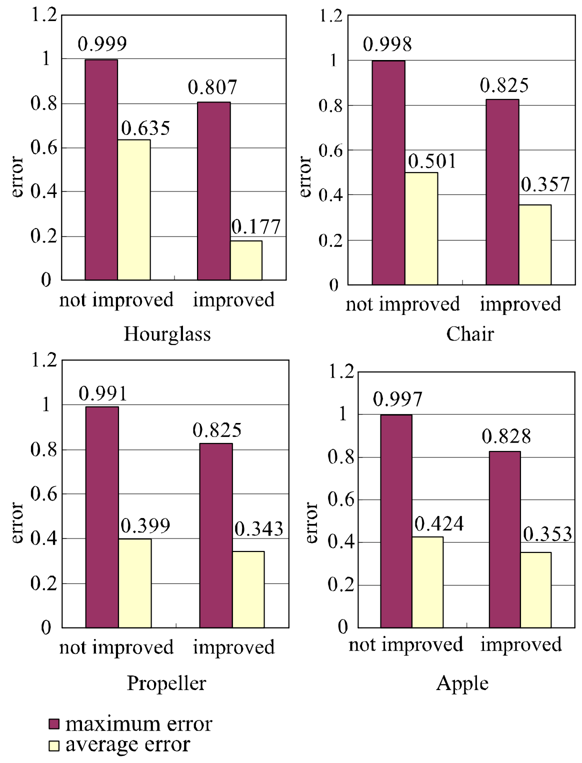


Figure 13. Comparison of accuracy of generated voxel data

Figure 13 shows that our method reduces the average error by 0.05-0.45. Similarly, the maximum error decreases. When the shape has complex concavity and convexity, the average error is reduced more than that for a simple shape by the process of improving the accuracy, as shown in Figure 13. For example, the average errors for the hourglass and the chair are reduced more than the average errors for the other experimental shapes. The reason is that the error between the generated voxel data and the input shape increases according to the increase of the shape's complexity and the sharpness of the dihedral angle.

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed a fast method for transforming the boundary representation model into the voxel model. We achieved this fast transformation method by using a rendering engine. The required amount of video memory is small in our method because it involves two-dimensional processing. It is possible to apply our method even when the resolution of the voxel data is high. We also proposed a method for improving the accuracy with a pixel shader. We hope that the performance of our method improves along with the rapid progress of the GPU. Our future work will focus on determining the optimal number of sampling points for improving the accuracy and on reducing the processing time of the method to improve the accuracy.

REFERENCES

[Bru90] P. Brunet, and I. Navazo :"Solid Representation and Operation Using Extended Octree" ,ACM Transaction on Graphics , Vol.**9**,No.2,pp.171-197, (1990).

[Cg]CgUsersManual:http://developer.nvidia.com/object/cg_toolkit.html

[Don04] Z. Dong, W. Chen, H. Bao, H. Zhang, and Q. Peng : "Real-time Voxelization for Complex Polygonal Models", Computer Graphics and Applications, 12th Pacific Conference on (PG'04), p.43-50, (2004)

[Fan04] Z. Fan, W. Li, X. Wei, and A. Kaufman :"GPU-based Voxelization and its Application in Flow Modeling", ACM Workshop on General-Purpose Computing on Graphics Processors, (2004) http://www.cs.sunysb.edu/˜vislab/projects/gpgpu/Jellyfish.pdf

[Kas03] Kase, K., Teshima, Y., Usami, S., Ohmori, H., Teodosiu, C., and Makinouchi, A. :"Volume CAD", Volume Graphics 2003 Eurographics / IEEE TCVG Workshop Proceedings, I. Fujishiro, K. Mueller, A. Kaufman (eds.) in cooperation with ACM SIGGRAPH, Tokyo, pp.145-150, pp.173, (2003).

[Hsi05] H. Hsieh, Y. Lai, W. Tai, and S. Chang :"A Flexible 3D Slicer for Voxelization Using Graphics Hardware", Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, pp.285-288, (2005)

[Kob95] Kobori, K., Ishiguro, K. and Kutuwa, T. :"An Accurate Conversion Method from Boundary Representations to Octree Data" , ISCIE, Vol.**8**, No.3, pp.97-105,(1995).

[Nak05] Nakamura, N., Nishio, K. and Kobori, K. :"Automatic Generation of Boundary Representation Models from Binary Voxel Data", The Journal of The Institute of Image Information and Television Engineers,Vol.**59**,No.10,pp.1445-1453,(2005).

[Ono03] Onoue, K., and Nishita, T. :"Virtual Sandbox", Proceedings of IEEE 2003 Pacific Conference on Computer Graphics and Applications, pp. 252-259, (Oct. 2003).

[Yon96] Yonekawa, K., Kobori, K. and Kutuwa, T. :"A Geometric Modeler by Using Spatial-Partitioning Representations", IPSJ Transactions, Vol.**37**, No.1, pp.60-69, (1996).

[Yam02] Yamachi, H. and Shindo, Y. :"A Technique for Object and Collision Detection by Z-buffer", IPSJ Transactions, Vol.**43**, No.6, pp.1899-1909, (2002).

[Yam04] Yamamoto, O. :"Fast Computation of Delaunay Triangulation using Graphics Hardware", Official Journal of JSIAM, Vol.**14**, No.4, pp.235-266,(2004).