Interactive Stipple Rendering for Point Clouds

Naoyuki Awano
Osaka Institute of Technology
1-79-1, Kitayama
Hirakata, Osaka
Japan(573-0196)
awano@is.oit.ac.jp

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

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

ABSTRACT

Non-photorealistic rendering has been attracting attention in the field of computer graphics. A common approach to artistic rendering is using a shape model which utilizes mesh data. In recent years, the use of point clouds as shape models has increased due to the rapid development of 3D-scanners used to create them. Correspondingly, there has been an increase in the research on point clouds. We propose a stipple rendering method as a type of artistic rendering for point clouds, based on a hybrid image/object space. First, we eliminate hidden points based on an image space. Next, we apply a novel shading method to the visible points based on an image space. Lastly, we apply the above two results to the input point cloud. We implement the proposed method using a graphics processing unit to accomplish the interactive rendering. The experimental results show that we can achieve shading and shadowing interactively.

Keywords

Computer graphics, non-photorealistic rendering, stippling, point cloud, and graphics processing unit.

1. INTRODUCTION

Non-photorealistic rendering (NPR) has become a major focus for research in the field of computer graphics, because it is an effective conveyor of geometric features. Considerable artistic rendering has been proposed using a 3D-shape model utilizing mesh model [Zan04][Sat04][Lak00][Say06]. A mesh model is suggested because it has a topological data structure that can be used to extract features. However, the mesh model must first be constructed from point clouds before any suggested methods can be applied to it.

Over the past few years, point clouds have attracted attention as a new shape model, because they can be easily created using 3D-scanners, which have seen rapid development lately. Correspondingly, there has been an increase in the research on point clouds [Pau03][Pfi00] including NPR studies.

For example, Zakaria [Zak04] proposed a hybrid image/object space method of interactive silhouette rendering. It can also do stipple rendering or user-drawn strokes on point set surfaces. Runions [Run07] proposed a novel rendering method for point clouds. It creates stripes, called "ribbons" to achieve photorealistic or non-photorealistic representations. Furthermore, it achieves NPR interactive silhouette rendering utilizing ribbons only. Rosenthal [Ros08] proposed an image space rendering method using a graphics processing unit (GPU). It is mainly used for photorealistic representations, and optionally for non-photorealistic representations. In addition, it can render silhouettes on photorealistic representations, and accordingly enables conspicuous representation.

In this paper, we propose an NPR method of stippling using a point cloud without normal vectors. We selected stippling because it is suitable for point clouds, which consist of points only. First, we eliminate the hidden points based on the image space. Next, we apply a new shading method to the visible points based on the image space; moreover, we can control the degree of shading. In addition, we implement all of the methods using a GPU to accomplish interactive rendering.

2. STIPPLE RENDERING

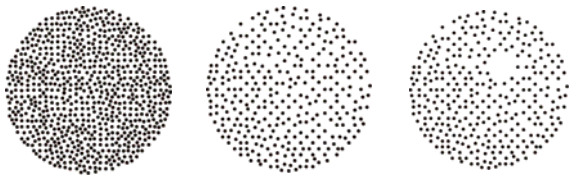
Stippling is an artistic rendering method which uses points only, and achieves shading by changing the density of the points. To get results for stippling

using a point cloud, we limit the background color to white and color all the points black. The input point cloud our method uses is evenly distributed without normal vectors and insufficiency.

Figure 1 shows the results of applying general shading to the mesh model. Figure 1(b) is the result of applying diffuse reflection to Figure 1(a). Figure 1(c) is the result of applying specular reflection to Figure 1(b). Typically, there are two steps to general shading; diffuse and specular reflection. Therefore, we apply our method, which also consists of diffuse and specular reflection, to the input point cloud. To get an effect similar to diffuse reflection, we thin out some of the points from Figure 2(a) as shown in Figure 2(b). In addition, we omit some local points for specular reflection as shown in Figure 2(c).

(a) (b) (c)

Figure 1. Shading; (a) original; (b) diffuse; (c) specular.



(a) (b) (c)

Figure 2. Shading using our proposed method.

Our method consists of four steps, which are outlined in Figure 3, and we implement all the methods using a GPU to accomplish interactive rendering.

Step 1) Creating a texture to eliminate the hidden points from the point set surfaces.

Step 2, 3) Creating textures for diffuse and specular reflections so that we can obtain similar effects.

Step 4) Applying the three textures applied to a point cloud and obtain the result.

2.1 Hidden points texture

In general, if we use the mesh data as input shapes, the back faces are eliminated from the front faces utilizing a Z-buffer. However, point clouds have points only, so the points on the back faces are not eliminated even if we apply a Z-buffer. Therefore, we apply following method so that we can apply the Z-buffer to eliminate the hidden points.

First, we place a texture plane at a viewpoint on the GPU at a size greater than the screen resolution. All points are then projected onto the texture plane. In Figure 4(a), the depth value of point A is 2, point B

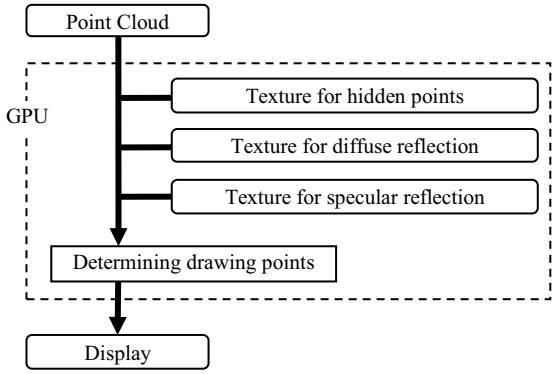


Figure 3. Outline of our method.

is 4, and point C is 3. So, each pixel has the smallest depth value of all the points which are points projected onto it. Next, as shown in Figure 4(b), in order to eliminate the hidden points in the texture plane, we store each depth value into the neighboring pixels, which also have the smallest depth value. After that, we apply the results of the texture plane to the point cloud; see Section 2.4.

However, there are cases where the hidden points on the back faces are not eliminated from the texture plane by the above process. In such cases, we repeat the above process until the hidden points are eliminated from the texture plane.

When B (Figure 4(b)) is eliminated from the texture plane, other points in high density parts of the texture plane also tend to be eliminated. As a result, too many points are eliminated from a high-density point cloud.

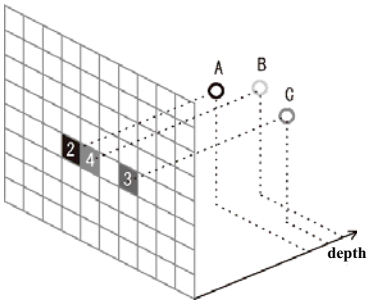


Figure 4(a). Projection onto a texture plane.

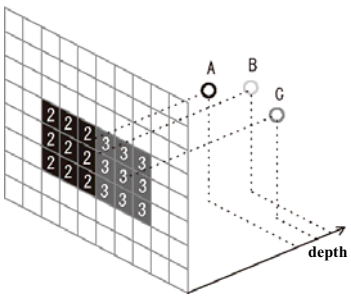


Figure 4(b). Storing the depth values into 8 neighboring pixels.

Hence, instead of storing the same depth value into the surrounding pixels, we store a slightly larger depth value, as shown Figure 5, so as to avoid cases where extra points are eliminated. In fact, if we assume that the maximum norm in all points is 1.0, we set the larger depth value is 0.05, which value was defined by our implementation. The final result of the hidden points texture is shown in Figure 6.

(a) Before changing. **(b) After changing.**

Figure 5. The centered pixel is projected by a point with depth value 3. In (a), the other pixels have the same depth value: 3. In (b), the other pixels have a slightly larger depth value of 6.

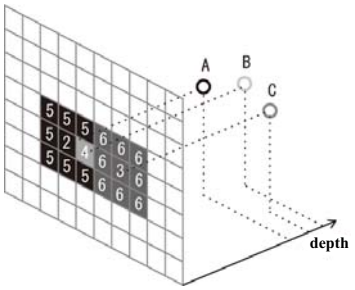


Figure 6. Improvement of storage.

2.2 Diffuse reflection texture

As shown in Figure 2(b), we thin out some points to represent diffuse reflection, and control the degree by changing the number of hidden points. In particular, we control the degree by changing the density of the points; a high-density part is low degree and a low-density part is high degree.

First, we place a texture plane at an illuminant on the GPU as a diffuse reflection texture. Next, all points are projected onto the texture plane as shown at the top of Figure 7. The density distribution of all the projected points on the texture plane is shown at the bottom of Figure 7. As a result of this, the lowest density part has the highest degree of diffuse reflection. Therefore, it is possible to achieve the effect of diffuse reflection by all points are projected only onto the texture plane. Additionally, we control the degree of diffuse reflection by changing the number of drawing points.

The idea is that each pixel on a texture has a hidden point, and we control the degree of diffuse reflection by changing texture size. In particular, after all visible points are projected onto the texture plane, each pixel has a minimum depth value, similar to the hidden points texture. The texture consists of multi-resolution textures so that we can control the degree of diffuse reflection, as shown in Figure 8(a). Then we define a full quadtree, with LEVELS 0-n.

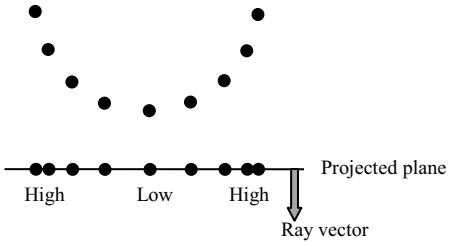
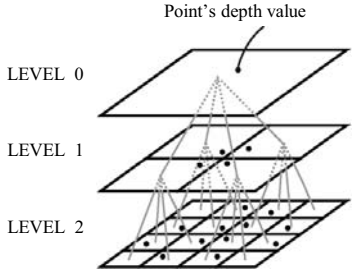


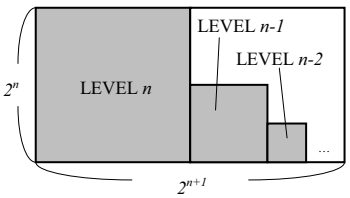
Figure 7. Density of projected points.

If we select a LEVEL, all stored points on the texture of the selected LEVEL are hidden. Moreover, the full quadtree is maintained on a texture so that we can effectively hold the textures on the GPU. In our method, we create the diffuse reflection texture, sized $2^{n+1} \times 2^n$, and store each LEVEL as shown in Figure 8(b). So, a pixel on LEVEL 1 has the minimum depth value of four pixels on LEVEL 2. Similarly, a pixel on LEVEL 2 has the minimum depth value of four pixels on LEVEL 3. Thus, all LEVELS can be created based on the texture of LEVEL n.

In particular, after the projection of all visible points onto the texture of LEVEL n, all of the other resolution textures are created in descending order,



(a) Multi-resolution texture.



(b) On a texture.

Figure 8. Full quadtree.

based on the texture of LEVEL n. So, e.g., a pixel (x, y) on the texture of LEVEL $(l-1)$ has the minimum depth value in (x', y') which is calculated by Formula (1), $(x'+1, y')$, $(x', y'+1)$, $(x'+1, y'+1)$.

$$(x', y') = (2(x - \sum_{k=l}^{n} 2^k), 2y) \quad \textbf{(1)}$$

2.3 Specular reflection texture

We refer to the Phong reflection model that is typically applied to a shape model. The degree of specular reflection is determined by the angle between the ray and normal vector of the surface. However, the input point cloud does not include normal vectors; therefore, instead of using normal vectors, we create parameter similar to the above angle.

First, we place a texture plane at an illuminant on the GPU as a specular reflection texture. Next, all points are projected onto the texture plane. Then, assume that all points have normal vectors as shown at the top of Figure 9. The angle between each normal vector and ray vector has the angle distribution shown at the bottom of Figure 9.

Note that the density distributions of the projected points in Figure 9 have a similar distribution to those in Figure 7. Therefore, we compare their distribution to achieve shading without a normal vector. In particular, we regard each of the density distributions on the projected plane to be the angle between a normal and ray vector. Similarly, if we replace the ray vector with the eye vector in Figure 9, we can regard the density distributions of each projected point with the angle between the normal and eye vector. Consequently, the differences of their distributions indicate the angle between the eye and ray vector.

In Figure 6, note that the depths based on point A were stored into 8 pixels; point C has 9 pixels. We have found that their numbers are almost proportional to the distributions on the hidden points texture.

Thus, we create a hidden points texture at an illuminant as a specular reflection texture on the GPU. Then, we determine the drawing points by referring to specular reflection texture and the hidden points texture; see the next section.

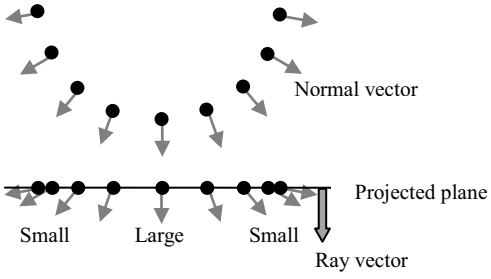


Figure 9. Angle between two vectors.

2.4 Determining drawing points

We determine the drawing points by utilizing all three textures. First, we determine the visible points by applying the results of the hidden points texture to the point cloud. Next, we apply the results of the diffuse reflection texture and specular reflection texture to the visible points.

2.4.1 Eliminating hidden points

To eliminate the hidden points, all points are projected again onto the hidden points texture, as shown in Figure 10, where the depth value of point A is 2, point B is 4, point C is 3, and point D is 7. It shows that point A is projected onto a pixel whose depth value is 2, and each adjacent pixel has a depth value of 4-5. Then, since there is a larger depth value than point A's depth value of 2, point A is drawn.

However, point D is projected onto a pixel whose depth value is 6, and each adjacent pixel has a depth value of 3-6. Then, since there is no larger depth value than point D's depth value of 7, point D is not drawn. Using this process, all points are projected onto the hidden points texture to determine whether they should be eliminated or not.

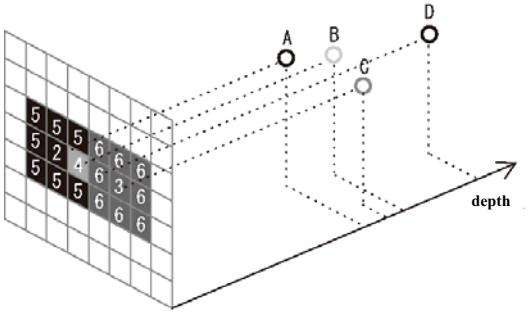


Figure 10. A result of hidden points texture.

2.4.2 Hiding points for diffuse reflection

We hide some of the points to represent diffuse reflection. This process is illustrated using Figure 11, where we can select any of the four hidden LEVEL (0-3).

We start by selecting hidden LEVEL 3. All visible points are projected onto the diffuse reflection texture of LEVEL 3, so as to represent the diffuse reflection with LEVEL 3. For example, if a point with depth value 4 is projected onto the pixel shaded with diagonal lines, the point is not drawn because it has the same depth value as that of the pixel. In contrast, when a point with depth value 5 is projected onto the same pixel, the point is drawn.

Next, when we select hidden LEVEL 2, all visible points are projected onto the diffuse reflection texture of LEVEL 3. For example, if a point with depth value 4 is projected onto the pixel shaded with diagonal lines, the pixel is related to a pixel such as the bold pixel in LEVEL 2 of Figure 11. Since, the bold pixel in LEVEL 2 has the depth value of 2, the point is drawn because its depth value is different from that of the pixel.

In case of selecting hidden LEVEL l, all points are first projected onto the location of LEVEL n. After that, when a point is projected onto a pixel (x, y), we refer to the pixel as (x', y'), computed by Formula (2). If the depth in the referring pixel is equal to the projected point's depth, the point is not drawn.

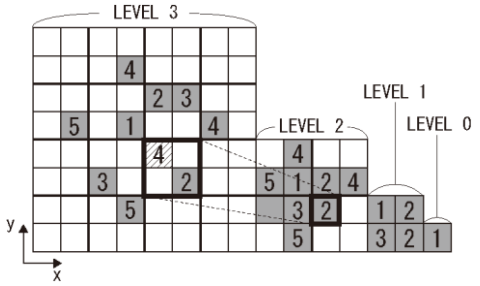


Figure 11. An example of diffuse reflection texture. (The numbers represent the depth value of each pixel.)

$$(x', y') = \begin{cases} (x, y) & (n = l) \\ \left((x / 2^{n-l}) + \sum_{k=l+1}^{n} 2^k, y / 2^{n-l} \right) & (n > l) \end{cases} \quad \textbf{(2)}$$

2.4.3 Hiding points for specular reflection

We hide some points to represent specular reflection in addition to diffuse reflection. First, all visible points are projected onto a specular reflection texture and a hidden points texture. As noted in Section 2.2, we count the pixels in each texture and calculate the difference between them. Then, we define a threshold ε to the difference so that we can control the degree of specular reflection; ε is the ratio of the difference to the number of pixels. The first iteration count is 9 and the second iteration count is 25. Next, all visible points that have a difference greater than ε are hidden.

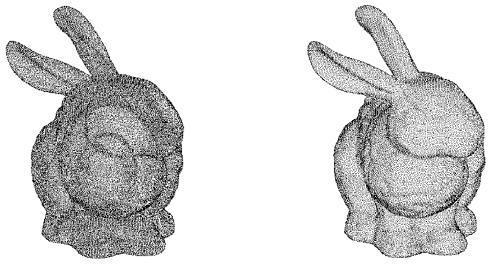
3. RESULTS

We conducted experiments and verified our method. We tested our method as shown in Table 1. The size of the diffuse reflection texture is 2048 × 1024; the hidden points texture and specular reflection texture are 1024 × 1024 each. Additionally, we adopted Cg for implementing our method on GPU, and assign coordinate value XYZ of all points to color value RGB in all three textures.

CPU	Core2 Duo 2.66 GHz
RAM	2.0 GB
GPU	GeForce 8800GTX
VRAM	768 MB

Table 1. Experimental environment.

Figure 12 shows the results of eliminating hidden points with 72,027 (Bunny). This indicates that we can eliminate the hidden points by repeating the process, if we cannot eliminate the hidden points the first time.



(a) once **(b) twice**

Figure 12. Eliminating hidden points.

Figure 13 shows the results of applying diffuse reflection to two point clouds: (a) and (b) are 542,199 (Oil pump); (c) and (d) are 152,807 (Chinese dragon). They indicate that we can achieve diffuse reflection as shown in Figure 2(b) and adjust the diffuse reflection by changing the LEVEL. Figure 14 shows the results of specular reflection on the shapes shown in Figure 13. It shows that the high-light of specular reflection, as shown in Figure 2(c), appears locally by using our shading method. Furthermore, they indicate that we can adjust the specular reflection by changing ε. Therefore, they indicate that we achieve shading by stippling.

Figure 15 shows another result of shading with 172,974(Armadillo). This indicates that our method can achieve shadowing as shown in the circle in Figure 15. Due to containing the eliminated hidden points in our shading method, our shading method has not been applied to the part in the shadow.

Figure 16 shows the results of processing speed. We have compared the implementation on a CPU against a GPU. It shows that on a GPU, the speed is 11 to 17 times faster than on a CPU. The reason is that our method can be implemented with an image space, and the GPU can perform in parallel with an image space. Thus, our method can achieve interactive rendering.

4. CONCLUSION

In this paper, we proposed a NPR method with stippling using point cloud without normal vectors.

First, we eliminated hidden points. Next, we applied a novel shading method consists of specular reflection and diffuse reflection. In addition, we implemented our method on GPU to accomplish interactive rendering.

In the future, we will refine our method so it can be applied to an unorganized point cloud.

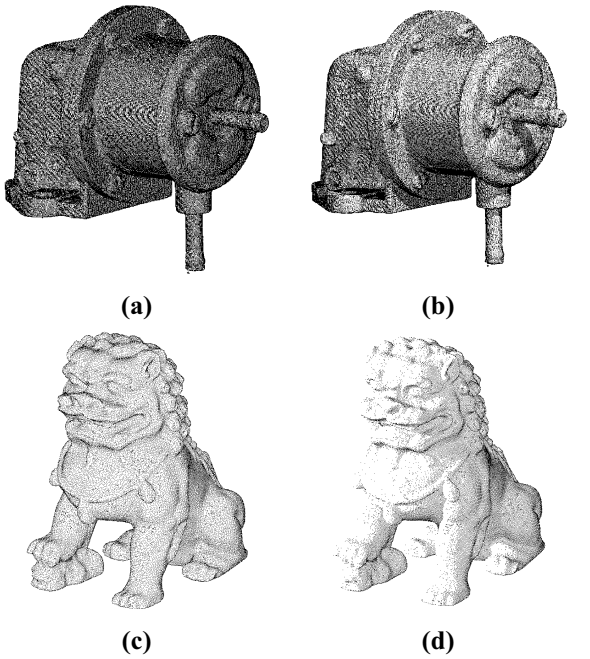


Figure 13. Results of after applying diffuse reflection.The hidden LEVEL of (a) and (c) are 7; (b) and (d) are 9.

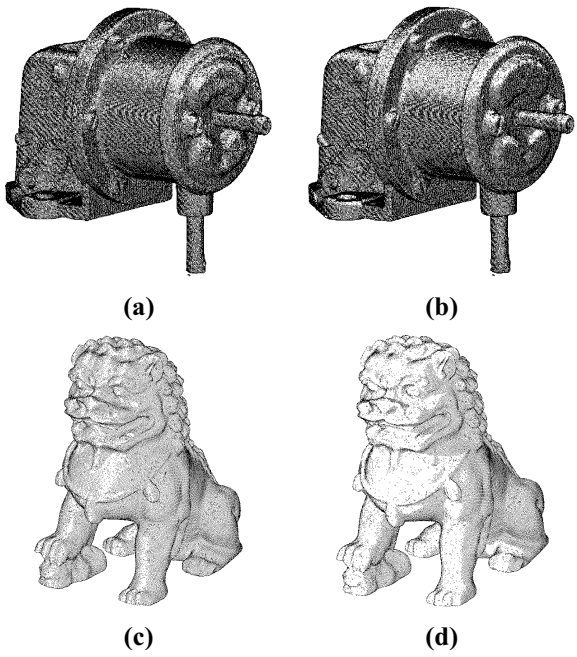


Figure 14. Results of applying specular reflection. ε of (a) and (c) is 0.3; (b) and (d) is 0.2.

5. REFERENCES

[Zan04] J. Zander, T. Isenberg, S. Schlechtweg, and T. Strothotte, High Quality Hatching, *Computer Graphics Forum*, 23(3), 2004, 421-430.

[Sat04] Y. Sato, T. Fujimoto, K. Muraoka, and N. Chiba, Stroke-Based Suibokuga-Like Rendering for Three Dimensional Geometric Models, *The Journal of the Society for Art and Science 2004*, *3(4)*, 2004, 224-234.

[Lak00] A. Lake, C. Marshall, M. Harris, and M. Blackstein, Stylized Rendering Techniques For Scalable Real-Time 3D Animation, *1st International Symposium on Non-Photorealistic Animation and Rendering*, 2000, 13-20.

[Say06] R. Sayeed, T. Howard, State of the Art Non-Photorealistic Rendering (NPR) Techniques, *EG UK Theory and Practice of Computer Graphics*, 2006, 1-10.

[Pau03] M. Pauly, R. Keiser, L.P. Kobbelt, and M. Gross, Shape Modeling with Point-Sampled Geometry, *Proceedings of SIGGRAPH 2003*, 2003, 641-650.

[Pfi00] H. Pfister, M. Zwicker, J.V. Baar, and M. Gross, Surfels: Surface Elements as Rendering Primitives, *Proceedings of SIGGRAPH 2000*, 2000, 335-342.

[Zak04] N. Zakaria and H.P. Seidel, Interactive Stylized Silhouette For Point-Sampled Geometry, *GRAPHITE 2004*, 2004, 242-249.

[Run07] A. Runions, F. Samavati, and P. Prusinkiewicz, Ribbons: A representation for point clouds, *The Visual Computer: International Journal of Computer Graphics*, *23*, 2007, 945-954.

[Ros08] P. Rosenthal and L. Linsen, Image-space Point Cloud Rendering, *Proceedings of Computer Graphics International 2008*, 2008, 136-143.

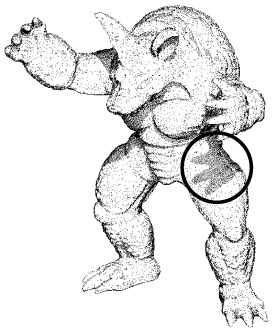


Figure 15. A result of shadowing.

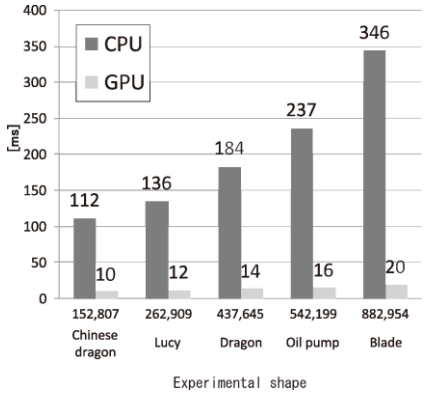


Figure 16. Processing speed.