2011 International Conference on Cyberworlds

Evaluation of a Technique for Collision and Object Detection with the Z-buffer in Cyber Space

Hidemi Yamachi, *Member, IEEE,* Yasuyuki Souma, Yasushi Kambayashi, *Member, IEEE*,
Yasuhiro Tsujimura, *Member, IEEE*, Tomoaki Iida
Nippon Institute of Technology
e-mail:yamachi@nit.ac.jp

Abstract— **We propose a new technique to detect objects and collisions of geometric objects in cyber space. This technique uses depth values of the Z-buffer in rendering scene. We use the orthographic projection for collision detection. Our method uses two depth value sets. One is obtained through rendering the cyber space from the sensor object toward a target point. This set does not have the depth values of the sensor object. Another one is obtained through rendering only the sensor object in the reverse direction. From these two depth value sets, we obtain the distance between the sensor object and others for each pixel. This technique requires only one or two rendering processes and it is independent from the complexity of the object's shape, deformation or motion. In this paper we evaluate the efficiency of this method on the GPUs we currently use.**

Keywords-component; Z-buffer; collision detection; object detection; Cyber Radar

I. INTRODUCTION

The problem of collision detection in virtual space has been intensively studied in various fields such as CAD, robotics, computational geometry, and computer graphics. There are a variety of algorithms for it. Most of them have the same feature. They calculate geometrical intersections between each object pair. In order to reduce the calculation cost, they divide the space or objects into convex polygons and eliminate obviously not colliding parts. Initially they have to construct bounding volume hierarchy trees, BVH. They work very well for rigid models that never change their shapes. Once the models change their shapes, however, they have to reconstruct the BVH. This procedure takes considerable time and space computational complexities. All those methods perform one collision test for every pair of objects. If there are several potentially colliding object pairs, they have to perform the collision tests for all of them.

In order to mitigate this situation, rendering based algorithms have been proposed. They exploit GPU's rendering power for collision detection. They don't have to construct any trees like BVH in order to specify prospective object pairs. They utilize the depth buffer values. They render potentially colliding object pairs and obtain depth values from the Z-buffer. Each object is rendered in the front surface and the back surface. All obtained depth values are kept and compared for detect intersection point detections.

Main contribution: In the previous paper, we proposed an algorithm for the collision detection of geometric objects, named *Cyber Radar*[1]. The algorithm uses two sets of depth values obtained through rendering. It uses a viewing volume of orthographic projection which is defined as the *target area* where an object passes through. We call the moving object the *sensor object*. The sensor object is the object we want to detect its collision. At first, we set the view point at the sensor object and render the scene toward the end of the viewing volume without the sensor object. If there is no depth value less than the maximum depth value, no other object in the target area and no collision will occur. If this is not the case, it sets the view point at the opposite position and renders only the sensor object, without any other objects. From those two sets of depth values, we can find the distance between the sensor object and other objects in the target area.

Cyber Radar takes only two renderings for one detection test. Different from other methods, it detects multiple collisions at once without selecting any object pair. Then *Cyber Radar* performs collision detection almost $O(n)$ time, where n is the number of polygons. *Cyber Radar* is also independent from the complexity of the object's shape, deformation or motion. In addition to these advantages, the implementation is simple and easy.

Through our recent survey of the collision detection problem, we found that the advantage of *Cyber Radar* is still clear in comparison with other methods. In this paper, we describe the details of our method and show the efficiency of our method through some experiments on newly developed GPUs.

Organization: The rest of this paper is organized as follows. In the next section, we briefly review related works and categorize them. In section 3, we describe the details of our object detection algorithm. We show the efficiency of our method through numerical experiments in section 4. We conclude our discussions in section 5.

II. RELATED WORKS

Since the 1980s, much research has been conducted to solve the collision detection problem. There are two basic ideas for collision detection algorithms. The first one is called object space culling [2]. It divides space and culls subspaces where no collision occurs. The second one is called image space interference computation. It utilizes GPU to obtain depth values [3].

Object space culling techniques can be further categorized into two types. The first type of techniques divides whole space into cells and keeps positions of objects using hierarchical trees, such as octree [4], BSPtree [5] and *k*-d trees [6]. The second type of techniques uses bounding volume hierarchies (BVH) which consist of convex hulls such as spheres or boxes. Convex hulls hierarchically cover an object from the whole to each part. AABB [7][8], Bounding Sphere [9], OBB [10], *k*-DOPs [11] are well known and used for collision detection. Because those techniques detect collisions only one pair of objects at once, they have $O(n^2)$ computational complexity for n objects. All those techniques take two phases, the broad phase and the narrow phase. The broad phase eliminates obviously not colliding part of space or objects. Then the narrow phase is performed to test geometrical intersections for each polygon pair in the prospective space. Even if space partitioning or BVH indicates some collisions, we sometime find that there are no collisions when we work on each polygon pair in the narrow phase. In addition to that, deformation of an object makes BVH have to be reconstructed. The reconstruction cost is not negligible. In order to ameliorate this problem, many methods have been proposed. Cohen *et al*. [12] utilize temporal coherence to reduce the number of interference tests. Govindaraju *et al.* [13] proposed a collision culling algorithm using GPU. Lauterbach *et al.* [14] proposed a method of parallel BVH construction method using GPU.

Image space interference computation employs the Z-buffer or the stencil buffer for collision detection. Rossignac *et al.* [15] used the Z-buffer and the stencil buffer. They use scan-conversion technique. Rays are cast toward points on clipping planes and the number of pass through the surface of solid plane is counted while clipping planes are being moved. Shinya and Fogue [3] propose a method that makes a list of the Z-buffer values for each pixel with IDs of objects. Each list includes minimum and maximum depth values of all objects at each pixel. The lists are sorted in order of depth value. If each object's minimum and maximum depth values are not adjacent, there is intersection at the pixel. Objects are rendered one by one and every time the Z-buffer has to be read. It requires significant overhead. Myszkowski *et al.* [16] used the Z-buffer and the stencil buffer. Their method deals with convex and concave objects. This method works on pre-sorted objects. It casts ray to objects and uses the stencil buffer to count the ray enter and exit objects before the ray reaches a surface point on another object of interest. Their algorithm works on only a pair of objects. Knott and Pai [17] also used the Z-buffer and the stencil buffer. Their method draws edges of all objects first. A ray is cast toward objects and when the ray strikes an edge, the number of back-facing and front-facing polygons between the point of the edge and the origin of the ray at the view point. If the numbers of front-facing and back-facing polygons are not equal, the edge point is in an object. This algorithm can not handle

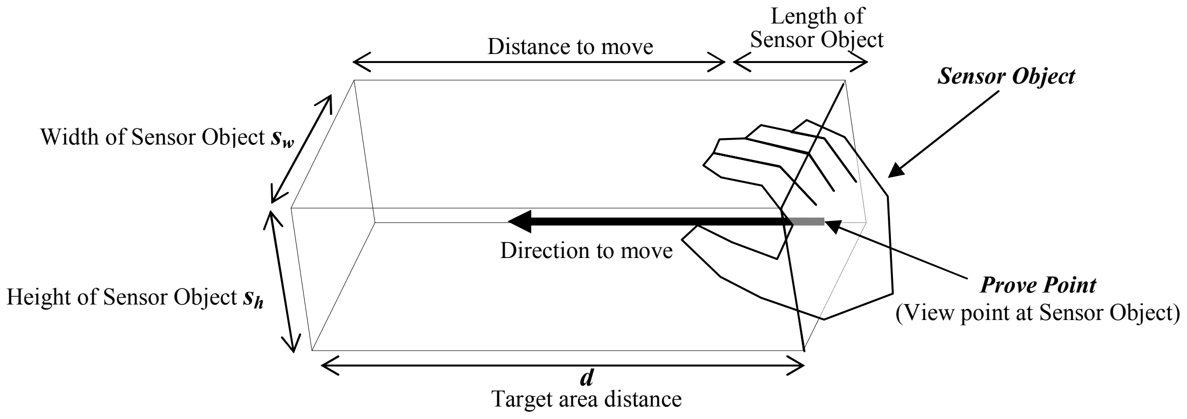
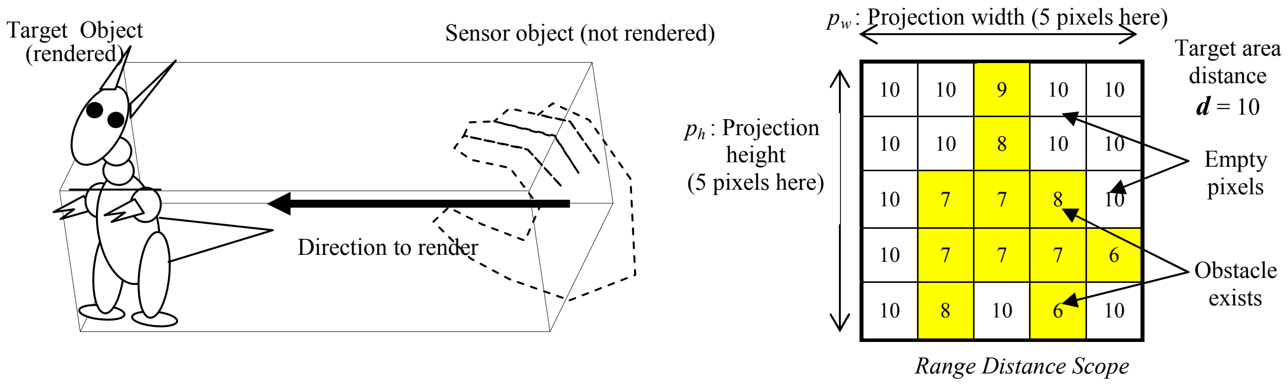


Figure 1. Sensor object and target area


Target Object (rendered)
Sensor object (not rendered)
Direction to render
p_w: Projection width (5 pixels here)
p_h: Projection height (5 pixels here)
Target area distance d = 10
Empty pixels
Obstacle exists
Range Distance Scope

10	10	9	10	10
10	10	8	10	10
10	7	7	8	10
10	7	7	7	6
10	8	10	6	10

Figure 2. Rendering for object detection phase

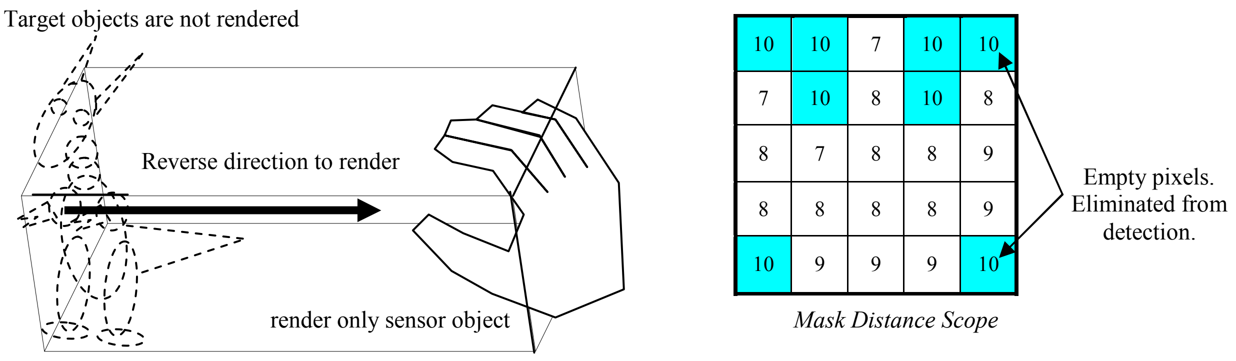

Target objects are not rendered
Reverse direction to render
render only sensor object
Empty pixels. Eliminated from detection.
Mask Distance Scope

Figure 3. Rendering for collision detection phase

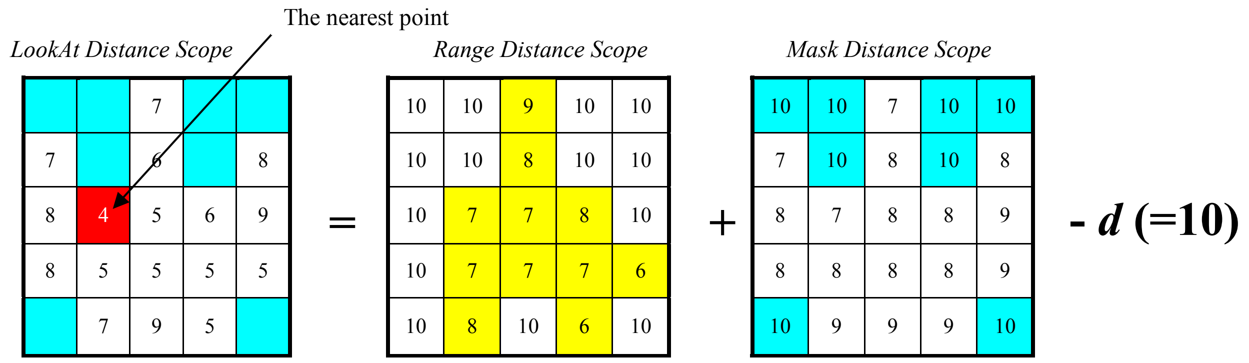


Figure 4. Calculation of *LookAt Distance Scope*

occluded edges. Usually image space techniques have drawbacks on reading GPU buffers. But by using edges of objects, they successfully reduce the overhead.

III. ALGORITHM

A. *Basic Concept*

We assume that objects don't collide in their initial state. We also assume objects are not in a situation that their convex hulls have no intersections between them such as linked rings or engaged gears. With these restrictions, we can assume a collision occurs only in the volume in which an object moves through and only on surfaces which face toward the moving direction.

Based on these assumptions, it is reasonable to restrict our consideration in the volume as the target area of the collision detection. The collision occurs only in the moving direction of the object.

Then we define the orthogonal projection view volume that has the width and the height of the sensor object and the depth same as the distance to move.

Cyber Radar detects collision through two phases. In the first phase, *Cyber Radar* renders the scene from the sensor object toward the direction to move without the sensor object, and it obtains the depth value set. If there is no pixel that has a value less than the maximum depth value, no collision will occur by the motion of the sensor object.

Otherwise, if there are some pixels that have values less than the maximum depth value, the view point is set at the other side of the target area and only the sensor object is rendered. Then the depth value set of the sensor object is obtained. Using these two depth value sets, collisions in the target area are detected.

We describe the details of these two phases in the following sections.

B. *Object Detection phase*

In order to detect other objects in the target area, the view point is set at the sensor object (Figure 1). We call this point the *prove point*. Then the orthogonal view volume, which has the depth that is the length of the sensor object plus the distance to move, and the height and width those are the same as the sensor object's, is defined as the target area. Then the scene it rendered without the sensor object. We call the obtained depth value set *Range Distance Scope*. If there are any pixels in the *Range Distance Scope* that have the depth values less than the target area distance ***d*** in Figure 1, there are some objects in the target area and we must perform collision detections.

Range Distance Scope (denoted by **R**) is defined as follows:

p_w , p_h : projection width and height

$\mathbf{R} \in \{R_{ij}\}\ \ i = 1,2,\cdots,p_w,\ \ j = 1,2,\cdots,p_h$

R_{ij} : depth value of pixel (i,j)

The result of object detection is given under the following conditions.

Object(s) exists $\quad min(\mathbf{R}) < \boldsymbol{d} \quad (1)$

No object exists $\quad min(\mathbf{R}) = \boldsymbol{d} \quad (2)$

$\boldsymbol{d}$: Target area distance

The algorithm of the object detection is as follows:

```
1. Set up the rendering conditions (Figure 1).
     View point at the Prove point.
    Render toward the direction the sensor object moves.
    $s_w, s_h$ := Width and height of the target area
    $d$ := Target area distance
    $p_w, p_h$ := Projection width and height
2. Render the scene without the sensor object (Figure 2).
3. Obtain R (Range Distance Scope).
4. if min($\mathbf{R}$) = $d$
      return false  (no object)
   else
      return true   (objects are detected).
```

Through this algorithm, multiple objects can be detected simultaneously. In order to distinguish each detected object, we can employ the color code method. The color code method assigns an ID number to each object in prior and the each ID is used as the color of each object. At the step 2, we render objects in the scene with the color of their ID number. By reading the object detected pixels in frame buffer, we can identify each objects by the color number.

C. Collision Detection phase

A collision occurs on the surface of a sensor object and other objects facing each other in a target area. This means that *Cyber Radar* is able to handle non convex objects in the direction of detection.

In the object detection phase, when some objects are detected in the target area, the collision detection is performed in the collision detection phase. The view point is set at the other side of the target area with the direction toward the sensor object. Then only the sensor object is rendered to obtain the depth value set. We named the depth value set *Mask Distance Scope*. With these two depth value sets, we can obtain the distances between the sensor object and other objects. We named this distance value set *LookAt Distance Scope.*

Mask Distance Scope and *LookAt Distance Scope* are defined below:

$\mathbf{M}$: *Mask Distance Scope*

$\mathbf{L}$: *LookAt Distance Scope*

$\mathbf{M} \in \{M_{ij}\} \; i = 1,2,\cdots,p_w, \;\; j = 1,2,\cdots,p_h$

M_{ij} : depth value of pixel (i,j)

$\mathbf{L} \in \{L_{ij}\} \; i = 1,2,\cdots,p_w, \;\; j = 1,2,\cdots,p_h$

L_{ij} : depth value of pixel (i,j)

The distance between prove point and a target object is

$$\boldsymbol{d} - \boldsymbol{R_{ij}}$$

The distance from the end of the target area to the sensor object is

$$\boldsymbol{d} - \boldsymbol{M_{ij}}$$

Using these values, the pixel values of *LookAt Distance Scope* are defined as follows:

$$\begin{aligned} \boldsymbol{L_{ij}} &= \boldsymbol{d} - (\boldsymbol{d} - \boldsymbol{R_{ij}}) - (\boldsymbol{d} - \boldsymbol{M_{ij}}) \\ &= \boldsymbol{R_{ij}} + \boldsymbol{M_{ij}} - \boldsymbol{d} \qquad (3) \end{aligned}$$

If $min(\mathbf{L})$, the minimum value of $\mathbf{L}$, is less than the distance to move, the collision will occur at the pixel. The time of collision is calculated from $min(\mathbf{L})$ and the velocity of objects.

If some objects are detected in the object detection phase, the collision detection phase is described below.

```
1. Set the rendering conditions (Figure 3).
      Set the view point at the end of the target area.
      Render toward the direction of the sensor object.
2. Render the scene only the sensor object.
3. Obtain M (Mask Distance Scope).
      Swap right and left pixel values.
4. Calculate L (LookAt Distance Scope).
      $\mathbf{L} := \mathbf{R} + \mathbf{M} - d$
5. if min($\mathbf{L}$) > Distance to move
      return false (no collision)
   else
      return true (collision is detected).
```

In order to obtain the *Mask Distance Scope*, the view point is set at the end of the target area with the direction toward the sensor object at the step 1. Then only the sensor object is rendered and obtained the depth values at the step 2. Upon obtaining the *Mask Distance Scope* at the step 3, the depth values of right and left pixels are swapped because their direction is reverse toward the *Range Distance Scope.* If a pixel of *Mask Distance Scope* has the value that is equal to $\boldsymbol{d}$, we eliminate the pixel from detection because the pixel does not collide.

At the step 4, the calculation of the *LookAt Distance Scope* is performed as follows:

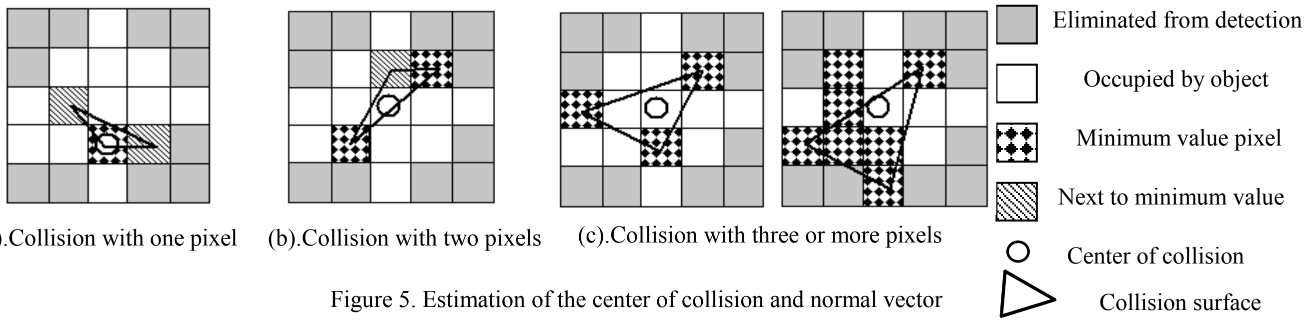
```
repeat  $i$ := 0 to $p_w - 1$, $j$ := 0 to $p_h - 1$
    if $M_{ij} = d$
        $L_{ij}$ := Value to eliminate
    else
        $L_{ij} := R_{ij} + M_{ij} - d$
```

An example of calculating *LookAt Distance Scope* is shown in Figure 4.

Step 4 eliminates all the pixels whose depth values are equal to $\boldsymbol{d}$. Through this procedure, *Cyber Radar* handles deformable models such as a set of discrete parts or an object that has some holes, as far as the features reflect on the *Mask Distance Scope.*

D. Estimation of Contact Points and Normal Vector of Collision Surface

In order to estimate the center of a collision and the normal vector of the collision surface, we have to consider three cases based on the number of pixels on the collision surface (Figure 5).



(a).Collision with one pixel (b).Collision with two pixels (c).Collision with three or more pixels

Figure 5. Estimation of the center of collision and normal vector

In the case of one colliding pixel, the center of the pixel is the center of collision. Other two pixels are chosen to form a triangle. The two pixels are chosen that have the values next to the minimum values (Figure 5 (a)).

In the case of two pixels, the center of the collision is the midpoint of the two pixels. The third pixel that has the value next to the minimum value is chosen to estimate the normal vector (Figure 5 (b)).

In the case of three pixels or more, one way to estimate the center of the collision is to calculate the center of gravity of the circumscribed polygons of the pixels. The normal vector of collision surface is calculated from three pixels of them (Figure 5 (c)).

E. Reuse of Mask Distance Scope and Dynamic Definition of Target Area Size

Cyber Radar needs to render the scene twice for collision detection. But when the shape of the sensor object reflects on the *Mask Distance Scope* does not change, we can reuse the *Mask Distance Scope* again. Also if the sensor object is a sphere, its *Mask Distance Scope* never changes and we can use it repeatedly.

When a sensor object changes its shape or posture, we must redefine the space size of view volume and regenerate the *Mask Distance Scope*. To define the precise width and height of view volume, the *Mask Distance Scope* is used as follows:

```
if Shape or posture changed
    MaskResetFlag := true
………………..
if MaskResetFlag = true then begin
    if $s_w$, $s_h$ = Max width, height of sensor object
        $s_w$, $s_h$:= Get precise size from current
                          Mask Distance Scope
        MaskResetFlag := false
    else
        $s_w$, $s_h$:= Max size
    Create new Mask Distance Scope with $s_w$, $s_h$
end
```

The maximum width and height of the sensor object are predefined. After the *MaskResetFlag* is turned on, s_w and s_h are set to this maximum value and *Mask Distance Scope* is obtained at the first time of detection. At the next time, through the min-max test of *Mask Distance Scope*, s_w and s_h are set to the precise sizes.

IV. NUMERICAL EXPERIMENTS

Based on the algorithm of *Cyber Radar* depicted in the former section, we can expect that the collision detection time by *Cyber Radar* does not depend on the number of objects with the same numbers of polygons. Also we can expect that the time is proportional to the number of collision detections.

The higher the resolution, the higher the precision we obtain the results of collision detection. However, it results in the increase of the depth buffer read back time. It is said that the reading buffers from GPU memory to main memory is much slower operation than that from main memory to GPU. This could be one of the bottle necks of the rendering based techniques [18], [19], [20].

In order to investigate these problems, we have conducted experiments as follows. In order to compare the performance under different conditions, we used three sets of PCs for the experiments listed in Table 1. We have implemented our algorithm for the experiments using Visual C++ 2008 with OpenGL and GLUT.

TABLE I. EXPERIMENT CONDITIONS

model	CPU	GPU	RAM	OS
PC1	Intel Core 2 Duo 1.6GHz	NVIDIA GeForce 320M	4GB	Windows 7
PC2	Intel Core 2 Quad 2.40GHz	NVIDIA GeForce 8800 GTS	2GB	Windows Vista
PC3	Intel Xeon 3500 2.66GHz	NVIDIA GeForce GT 120	8GB	Windows 7

A. Number of collision detections per frame

We arranged 64 boxes and 50 teapots that have 6,256 polygons each, in a large box. The total number of polygons is 313,190 (Figure 10 (a)). We set the 10 to 50 teapots as the sensor objects, and measured the collision detection time per frame. The resolution is set to 200x200. Figure 6 shows the results. We can observe that the collision detection time is almost proportional to the number of collision detections.

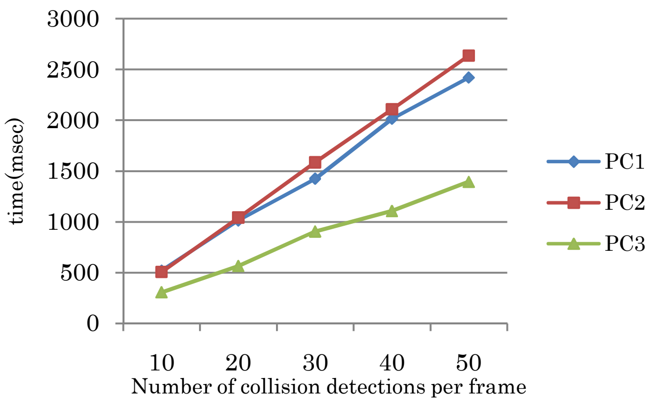


Figure 6. Number of Collision detection

B. *Resolution and polygons*

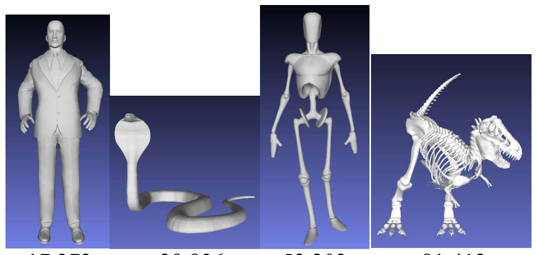
We measured the collision detection time for different resolutions 300x300 through 700x700 in the cyber space in which 64 mannequins located in a large polyhedron (Figure 10 (b)). For each case, one sensor object, (a) through (d) in Figure 7, is located. Each sensor object moves in the cyber space with collision detection. The process of collision detection is performed in a frame. According to the results in Figure 8, the difference of resolutions does not show remarkable difference of execution time period. Although the number of pixels in 700x700 is about 5.5 times as many as that of 300x300, the time of collision detection increases only about 28% in maximum and about 20% in average.

At the same time, the relation between number of polygons and time of collision detection is shown in Figure 9. It shows that the time of collision detection is about proportional to the number of polygons.

V. Conclusions

We propose a novel technique to detect objects and collisions of geometric objects in cyber space. This technique uses depth values of the Z-buffer in rendering scene. We use the orthographic projection for collision detection. Our method uses two depth value sets. One is obtained through rendering the cyber space from the sensor object toward a target point. This set does not have the depth values of the sensor object. The other one is obtained through rendering only the sensor object in the reverse direction. From these two depth value sets, we obtain the distance between the sensor object and others for each pixel. This algorithm is independent from the complexity of the shapes of the objects; the deformation of the objects or the motions of the objects do not increase the complexity either.

We have conducted experiments to investigate the difference of computational complexity for the number of collision detections per frame, resolutions and the number of polygons. The results show that the time of collision detection is proportional to the number of collision detection and polygons. Higher resolutions for rendering based



Polygons	17,372	29,836	52,202	81,412
	Human	Snake	Robot	Dinosaur
	(a)	(b)	(c)	(d)

Figure 7. Sensor Objects

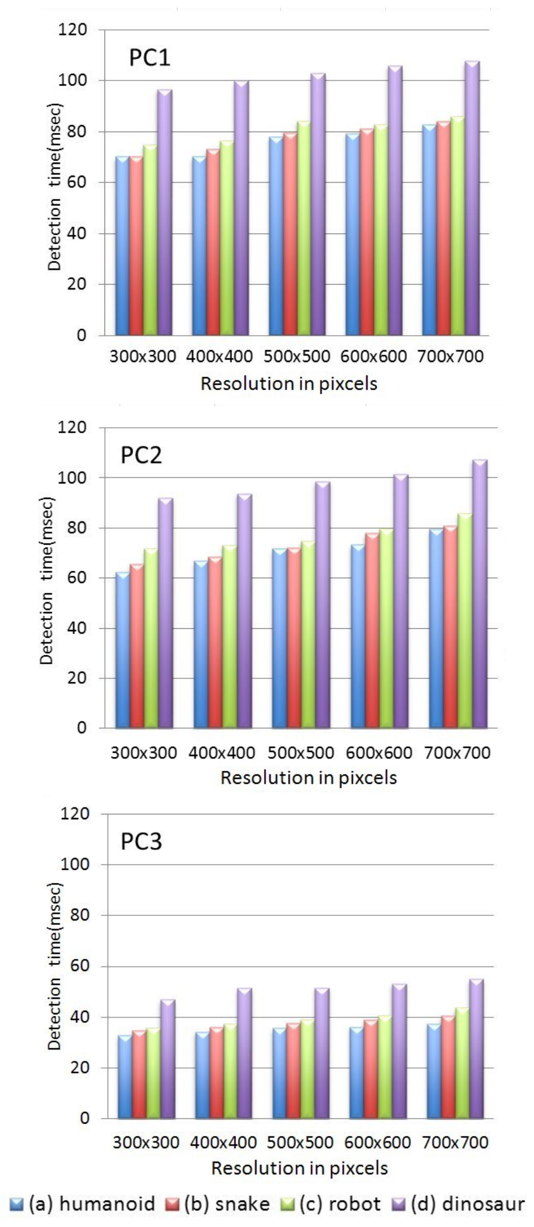


Figure 8. Difference of detection time by resolutions and polygons

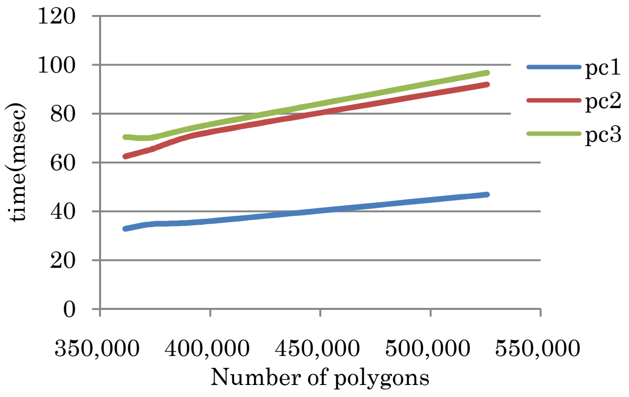


Figure 9. Difference of detection time by polygons

collision detections is said to be bottle neck. But more than 5 times as many pixels provide only about 20% increase of collision detection time.

Cyber Radar is easier to implement than space partitioning methods or bounding volume based methods.

The space complexity of *Cyber Radar* is about 3 times of the Z buffer size which is defined by the resolution for collision detection. The number of objects or polygons does not affect on the space complexity.

However, even *Cyber Radar* has some problems. In the case of rotation, *Cyber Radar* has to detect collision two times in both sides of the axis of rotation toward the orthogonal direction of the axis.

As we mentioned before, rendering based methods are restricted their detection precision within the pixel size. In order to ameliorate this problem, multiple collision detection with *Cyber Radar* could be effective. When a collision is detected, we can focus on the colliding area with the same resolution, and then we can perform collision the detection again for better precision. We also can exploit the color code method to identify the polygon and perform geometrical intersection test between colliding polygons.

We need to apply *Cyber Radar* for a variety of types of practical cases and to develop effective ways to use it.

ACKNOWLEDGMENT

This work is partially supported by Japan Society for Promotion of Science (JSPS), with the basic research program (C) (No. 23510178), Grant-in-Aid for Scientific Research.

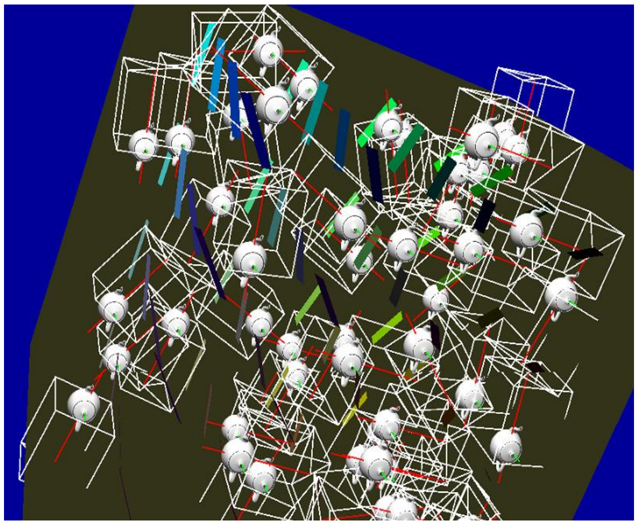
REFERENCES

[1] H. Yamachi and Y. Shondo, "A Technique for Object and Collision Detection by Z-buffer," Transactions of Information Processing Society of Japan vol. 43(6), pp. 1899-1909, 2002 (in japanese).

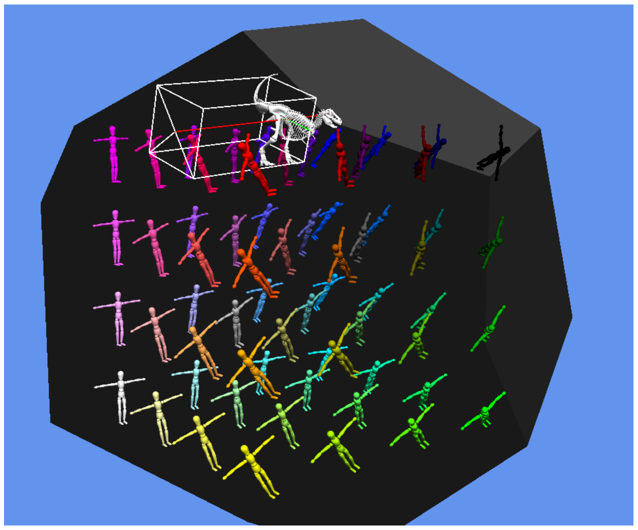
[2] N.K. Govindaraju, M.C. Lin and D. Manocha, "Fast and Reliable Collision Culling using Graphics Hardware," IEEE Transaction on Visualization and Computer Graphics, vol. 12(2), pp. 143-154, 2006.

[3] M. Shinya and M. C. Forgue, "Interference detection through rasterization," Journal of Visualization and Computer Animation, vol. 2(4), pp131-134, 1991.

[4] H. Samet, "Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods," Addison Wesley, 1989.



(a) Number of collisions



(b) Resolutions and polygon numbers

Figure 10. View of Experiments

[5] B. Naylor, J.A. Amatodes, and W. Thibault, "Merging BSP trees yields polyhedral set operations, " Computer Graphics (SIGGRAPH '90 Proc.), vol. 24, pp.115-124, 1990.

[6] M. Held, J.T. Klosowski and J.S.B. Mitchell, "Evaluation of collision detection methods for virtual reality flythroughs," Proc. 7th Canada Conf. Comput. Geom., pp.205-210, 1995.

[7] M. Ponamgi, D. Manocha, and M. Lin. Incremental algorithms for collision detection between solid models. IEEE Transactions on Visualization and Computer Graphics, vol.3(1), pp.51-67, 1997.

[8] G. van den Bergen, "Efficient collision detection of complex deformable models using aabb trees," Journal of Graphics Tools, vol.2(4), pp.1-13, 1997.

[9] P.M. Hubbard, "Approximating Polyhedra with Spheres for Time-Critical Collision Detection", ACM Transactions on Graphics, vol. 15(3), pp.179-210, 1996.

[10] S. Gottschalk, M. Lin, D. Manocha, "OBBTree: A Hierarchical Structure for Rapid Interference Detection", Proc. of ACM Siggraph '96, pp. 171-180, 1996.

[11] G. van den Bergen, "Efficient Collision Detection of Complex Deformable Models using AABB Trees." Journal of Graphics Tools, vol.2(4), pp.1-14, 1997.

[12] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi, "I-collide: An interactive and exact collision detection system for large-scale environments. In Proc. of ACM Interactive 3D Graphics Conference, pp.189-196, 1995.

[13] N. Govindaraju, S. Redon, M. Lin and D. Manocha, "CULLIDE:Interactive collision detection between complex models in large environments using graphics hardware," Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware, pp.25-32, 2003.

[14] C.Lauterbach, M. Garland, S. Sengupta, D. Luebke, D. Manocha, "Fast BVH Construction on GPUs," Computer Graphics Forum, vol.28(2), pp.375–384, 2009.

[15] J. Rossignac, A. Megahed and B.O. Schneider, "Interactive inspection of solids: Cross-sections and interferences", Proc. Computer Graphics (SIGGRAPH 92), vol.26, pp.353-360, 1992.

[16] K. Myszkowski, O. G. Okunev and T. L. Kunii, "Fast collision detection between complex solids using rasterizing graphics hardware," The Visual Computer, vol.11(9), pp.497-512, 1995.

[17] D.Knott and D.K. Pai, “CInDeR: Collision and Interference Detection in Real-Time Using Graphics Hardware,” Graphics Interface, pp.73-80, 2003.

[18] C.Funfzig, T.Ullrich,D.W.Fellner, “Hierarchical spherical distance fields for collision detection,” Computer Graphics and Applications, IEEE, vol.26(1), pp.64-74, 2006.

[19] C.Ericson, “Rel-Time Collision Detection,” Elsevier, pp.414-415, 2005.

[20] D.Kim, J.P.Heo, J,Huh, J.Kim and S.Yoon, “HPCCD: Hybrid Parallel Continuous Collision Detection using CPUs and GPUs”, Computer Graphics Forum, vol.28(7), pp1791-1800, 2009.