POSTER: Effective Object Sorting Technique for Developing 3D GUI Including Translucent Objects

Byungkwon Kang
Samsung Electronics
416, Maetan-3Dong, Yeongtong-Gu,
443-742, Suwon-city, Gyeonggi-Do, Korea
bk76.kang@samsung.com

Sunghee Cho
Samsung Electronics
416, Maetan-3Dong, Yeongtong-Gu,
443-742, Suwon-city, Gyeonggi-Do, Korea
shane.cho@samsung.com

ABSTRACT

In developing a user interface, it is very important to provide usability, intuitiveness and convenience. To accomplish these factors, 3D user interface can be one good solution. Recently, many kinds of embedded devices have been developed and utilized for various areas of human life; for example, Digital TVs, game consoles, MP3 players, etc. A GUI makes these devices easier to use and more effective. But, their poor hardware performances often prevent developing a high quality visual effect on their GUIs. A visual effect using alpha blended object is one of the typical examples of this kind that limits the development of 3D applications on an embedded hardware. In this paper, we propose an effective object sorting technique that can be used to develop a 3D GUI including many translucent objects.

Keywords

3D graphics, 3D GUI, Real-time rendering, Alpha blending, Object sorting, Translucent object.

1. INTRODUCTION

Alpha blending is a very useful technique that can be utilized in various areas of 3D graphics in order to generate more elegant visual effects. To composite colors of several translucent objects naturally, not only the rule for color composition but also the order of compositing objects must be selected carefully. These days, 3D graphics APIs such as OpenGL and Direct3D support blending of translucent objects' colors by using alpha channels very well. But, depth sorting which determines the order of composition for accurate blending of colors is not supported on current graphics hardware and APIs. Therefore, an effective sorting technique for correct composition of colors in an order of objects' distances is indispensable to high quality 3D graphics.

In this paper, we propose an effective object sorting technique that can be utilized for developing a 3D GUI including many translucent objects. In designing this technique, we defined some restrictions and conditions that are often found in developing a 3D GUI. These factors are very useful to simplify the method of determining spatial relations between 3D objects.

This paper is organized as follows. First, in Chapter 2, we introduce previous works about depth sorting methods and in Chapter 3, we summarize the preconditions and restrictions that are often found in developing 3D GUIs including translucent objects. In Chapter 4, we describe our depth sorting technique and summarize the performance and effectiveness in Chapter 5. Finally, we conclude this research in Chapter 6.

2. RELATED WORK

Depth ordering of 3D objects is a classic problem in various areas of computer graphics. There have been many types of research done to sort the primitives in object space [SBGS69, NNS72]. But their performances largely depended on the number of objects to be used. Therefore, there were many attempts to avoid the dependency on the number of objects [SBM94, KLNR97, SL98, WMS98, BOS04].

If the spatial positions of objects or polygons are not changed, a space partitioning data structure can be the most effective solution for ordering 3D objects. A binary space partition (BSP) of a set of objects is a recursive convex subdivision of space. It was designed to compute a visibility order among polygonal primitives used in graphics applications by

Fuchs et al. [HKN80]. The BSP tree has been a general solution and widely used for hidden surface removal, shadow generation, ray tracing, real-time 3D games, and so on. Most of the BSP algorithms work well only in static environments. But, traditional BSP algorithms were not suitable for a dynamic environment where the position of objects can be changed dynamically. Torres [Tor90] and Agarwal et al. [AGMV97] presented dynamic BSP data structures that are able to compute visibility ordering of polygons in a dynamic scene.

3. PRECONDITIONS AND RESTRICTIONS

In this chapter, we defined some preconditions and restrictions that are often found in implementing 3D GUIs. The preconditions and restrictions that will be considered in developing a 3D application focused on this research are as follows.

a. All objects in 3D space have a boundbox which has a shape of rectangular parallelepiped.

b. There is no intersection between each object.

c. It is developed on a dynamic environment that all spatial relations between objects change every time frame.

d. Object sorting is performed per each object using their boundbox.

All of these preconditions and restrictions are based on a premise that our research aims to develop a useful depth sorting method for 3D GUIs. Because our object sorting method is based on these conditions, our method is not suitable for some common 3D applications. But, the conditions described in this section are common features that can be found in most graphics user interfaces and they are very useful to simplify the object sorting process.

4. DESCRIPTION OF THE NEW SORTING TECHNIQUE

In this chapter, we describe a new sorting technique for accurate alpha blending proposed in this paper.

Finding overlapped objects

For the first step, find all objects overlapped with other ones. To make the order of composition of all objects in the scene, the spatial relations between object pairs must be determined. The process of finding overlapped objects in the eye coordination space is very simple. Because all boundboxes are shaped as a rectangular parallelepiped, up to 3 faces of a box are visible from the view point. Using these visible faces, we can construct a quad or hexagon using vertices on visible faces. After then, by applying a 2D overlapping test to these polygons we can easily find overlapping objects.

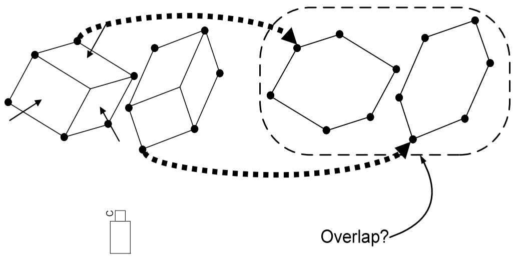


Figure 1. Projecting boundboxes into 2D polygons.

Constructing 2D polygon from boundbox

To construct a 2D polygon, we need to find the sequence of vertices that are constructing the contour of visible region of a boundbox. To achieve this process effectively, we designed an easy way to find the vertex sequence that is constructing the contour of visible faces of a boundbox.

At first, we assigned a fixed order of indexes to all faces and each vertices of a boundbox. As illustrated in Figure 2, all faces and vertices have uniquely and sequentially assigned indexes. And the boundboxes are transformed into the eye coordinates. After this transformation process, we can find visible faces from the camera easily by comparing 4th coefficient of plane equation of each transformed faces.

Third, we must find the sequence of vertices that are constructing the contour of visible faces. Because all faces and vertices have fixed indexes, we can find which vertices are included in the contour vertex list earlier on run-time when the visible faces are found. On run time, we can find the contour vertex list very quickly by making an index that is calculated by finding indexes of visible faces and searching its vertex list from the look up table.

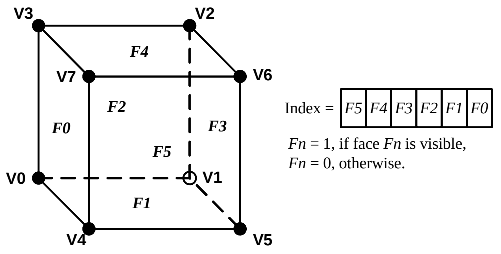


Figure 2. Making an index from visible faces.

After constructing projected polygon, the overlapping test can be performed very fast and effectively because this process is performed in 2D domain. There are many overlapping test techniques for 2D polygons and this test can be done easily [WMS98].

Finding front-back object

For each object which is overlapping with other objects, the spatial relation with the overlapped object must be discerned. To make an accurate alpha blended result, we must find which object is occluding the other object carefully and exactly. In this paper, we propose a simple and effective technique to find the spatial relation between overlapped objects based on the preconditions and restrictions described in Chapter 3.

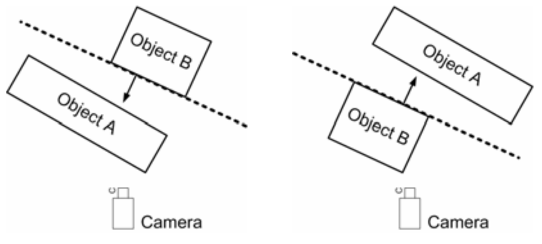
This technique is based on the restriction that all boundboxes must be shaped as a rectangular parallelepiped. From this restriction, we can determine the spatial relation between two boxes by finding the number of planes that all vertices of another object are placed in front of it. There are 3 cases between 2 non-intersecting boundboxes.

a. A places in front of only 1 plane of B.

b. A places in front of 2 or 3 planes of B.

c. There is no plane that A places in front of it.

Let's consider the first case. In this case, as illustrated in Figure 3, we can decide which object is the front object by discerning the spatial position of view point related to the selected plane. As illustrated in Figure 3(a), when the view point is placed in front of the plane, object A is placed in front of object B. Otherwise, when the view point is behind the plane as see in Figure 3(b), object B becomes the front object.



a) front case b) behind case

Figure 3. Spatial relation between selected plane and camera.

In cases except the first case, it is difficult to find front or back objects by using only 1 plane. But it is possible to find nearer object by comparing the nearest distances of vertices because the two objects are explicitly overlapped in the view of camera. After finding front and back objects between overlapped object pair, the front object is inserted to the front list of the back object and the back object is inserted to the back-list of the front object.

Rendering the objects in correct order

After front-back relations of all overlapped object pairs are discerned, the rendering order can be decided by using the result of discrimination. This process is performed based on a simple rule that any object cannot be rendered before its back-list becomes empty.

First of all, find an object having an empty back-list. The fact that an object has an empty back-list means the objects can be rendered immediately because there are no objects covered by the current object. The found object is inserted to the rendering queue and deleted from the back-list of its all front objects. If one of the front objects has an empty back-list, the object becomes a renderable object that can be rendered immediately. And then, the previous process is recursively performed in order to remove the newly selected object from the other object's back-list again. After finish this process for all object in the scene, the remaining objects which are not inserted yet are the non-overlapped objects. So these objects can be rendered without any ordering process.

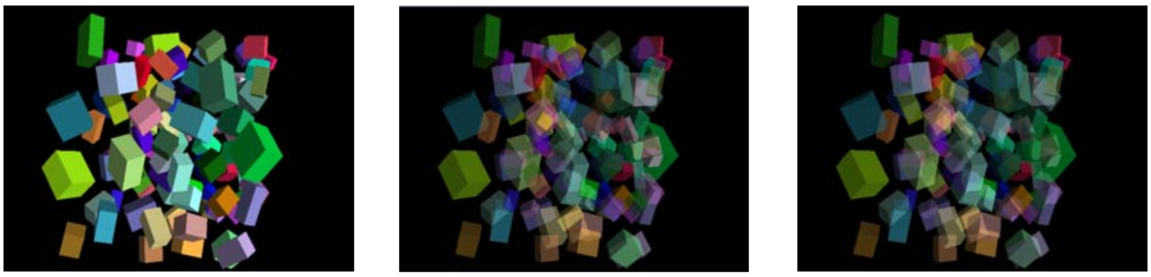
5. EXPERIMENTS AND RESULTS

To verify the performance and effectiveness of out technique, we performed several test on a desktop PC with a 2.13 GHz Intel Core2Duo CPU and 1 GB RAM. To test the performance, we generated random numbers of boxes and panels that are not intersected with any other boxes.

Table 1 shows the statistics of results of implemented applications using the proposed sorting technique. The test was done using several numbers of randomly-generated parallelopipedons and parallel panels. In Figure 4 and 5, we illustrated the results of rendered image under standard OpenGL environment. Figure 4 shows the comparison of not blended, incorrectly blended and correctly blended images. Figure 5 precisely shows the difference between correctly and incorrectly rendered results. As shown in Figure 5(b), the overlapped objects can be correctly rendered by using sorting technique proposed in this paper. In Table 1, we summarized the performances for 100 or 300 numbers of objects and we illustrated the performance for all number we considered in figure 6.

6. CONCLUSIONS

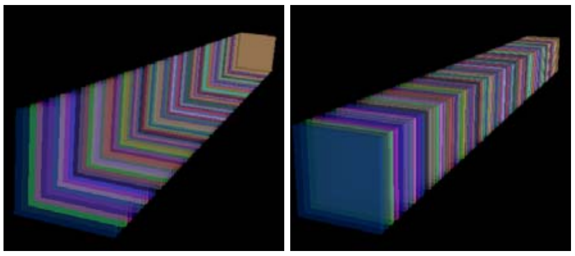
In this paper, we proposed a new depth sorting technique for accurate and fast alpha blending for developing 3D GUIs. In this technique, we designed several effective techniques to construct a projected 2D polygon from a boundbox and to determine the spatial relation between overlapped 3D objects. The proposed sorting technique is very easy to implement and it is possible to quickly arrange the depth order of each object.



a) without alpha blending b) blended without depth sorting c) correct result with depth sorting

Figure 4. Comparison of the correct and incorrect alpha blended results using 100 random boxes.

Although this technique is able to calculate the depth orders very quickly, the restrictions make it difficult to apply our technique for some application areas. In spite of the restrictions, this technique can be applied to various 3D applications in a limited hardware environment if there is not enough hardware performance.



a) incorrectly blended result b) correctly blended result

Figure 5. Comparison of the correct and incorrect alpha blended result using 100 parallel panels.

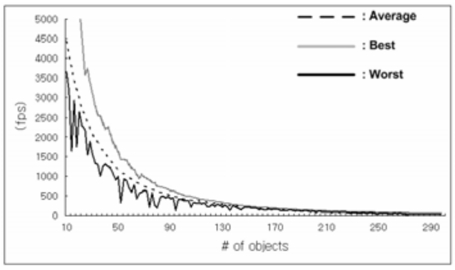


Figure 6. Rendering performance of proposed sorting technique for parallel panels.

	Random 100 boxes	Parallel 100 panels	Parallel 300 panels
Mean	293.5 fps	396.3 fps	55.3 fps
Best	1651.2 fps	476.3 fps	62.8 fps
Worst	95.9 fps	209.4 fps	49.6 fps

Table 1. Rendering statistics.

7. REFERENCES

[AGMV97] Agarwal P. K., Guibas L. J., Murai T. M. and Vitter J. S., Cylindrical static and kinetic binary space partitions. In *ACM Symposium on Computational Geometry*, pp. 39-48, 1997.

[BOS04]. Berg M., Overmars M. and Schwarzkopf O., Computing and verifying depth orders. In *SICOMP*, pp. 437-446, 2004.

[FDFH96] Foley J. D., Dam A., Feiner S. K. and Hughes J. H., Computer Graphics : Principles and Practice, 2nd edition, Addison-Wesley, 1996.

[HKN80] Huchs H., Kedem Z. and Naylor B., On visible surface generation by a priori tree structures. In *Proc. of ACM SIGGRAPH*, Vol. 14, No. 3, pp. 124-133, 1980.

[KLNR97] Karasick M. S., Lieber D., Nackman L. R. and Rajan V. T., Visualization of three-dimensional delaunay meshes. *Algorithmica*, Vol. 19, No. 1-2, pp. 114-128, 1997.

[NNS72] Newell M. E., Newell R.G. and Sancha T. L. A solution to the hidden surface problem. In *Proc. of ACM Nat. Mtg.*, 1972.

[SBGS69] Schumacker R., Brand B., Gilliland M. and Sharp W., Study for applying computer-generated images to visual generation. AFHRL-TR-69-74, US Air Force Human Resources Lab, Tech. Rep., 1969.

[SBM94] Stein C., Becker B. and Max N., Sorting and hardware assisted rendering for volume visualization. In *Symposium on Volume Visualization*, pp. 83-90, 1994.

[SL98] Snyder J. and Lengyel J., Visibility sorting and compositing without splitting for image layer decompositions. In *Proc. of ACM SIGGRAPH*, pp. 219-230, 1998.

[Tor90] Torres E., Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In *Proc. of Eurographics '90*, pp. 507-518, 1990.

[WMS98] Williams P. L., Max N. and Stein C., A high accuracy volume renderer for unstructured data. IEEE Trans. on Visualization and Computer Graphics, pp. 1-18, 1998