Object space versus image space
A comparison of image synthesis algorithms

T. van Rij

Computer Science/Department of Interactive Systems

Report CS-R9426 April 1994

Object Space versus Image Space
A Comparison of Image Synthesis Algorithms

T. van Rij
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
`tanja@cwi.nl`

Abstract

Images which are generated for graphic displays are generally stored in a frame buffer. This is a two dimensional array which represents the image in screen coordinates.
A three dimensional object needs a lot of calculations before it is transformed to such a frame buffer representation. An example of such a calculation is hidden surface removal. These calculations can either be done in 'object space' or 'image space'. Some people will argue that the image space hidden surface algorithms are the best and only way to go because of the efficiency of the algorithms, while others will rather use object space algorithms for their functionality. It can be argued that the best way is to combine the efficiency of the image space algorithms with the functionality of object space algorithms.
In this article the two approaches will be compared and their advantages and disadvantages will be laid out.

CR Subject Classification (1991): I.3.7: Three-dimensional graphics and realism - *visible line/surface algorithms, hidden line/surface removal*; I.3.6: Methodology and techniques - *graphics data structures and data types.*
Keywords & Phrases: Object space algorithms, Image space algorithms, rendering, hidden surface removal.

1. INTRODUCTION

The distinction between image space and object space was first made in [Sutherland et al., 74] for hidden surface removal algorithms. In that paper, calculations in object space are defined as being computations which compute 'exactly' what the image should be in the highest possible precision. Image space calculations calculate a value for each of the dots on the display screen. So object space algorithms work in continuous space, while image space algorithms work in discrete space.
List priority algorithms are in between object space and image space algorithms. These generally consist out of two phases. The first phase is performed in object space, however the second phase is done in image space.
In [Foley et al., 90] the difference between the two approaches is clearly explained by means of pseudo code.

Report CS-R9426
ISSN 0169-118X
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

image space approach:

```
for each pixel in the image do
   begin
      determine the object closest to the
      viewer that is pierced by the pro-
      jector through the pixel;
      draw the pixel in the appropriate
      colour;
   end
```

object space approach:

```
for each object in the world do
   begin
      determine those parts of the object
      whose view is unobstructed by other
      parts of it or any other object;
      draw those parts in the appropriate
      colour;
   end
```

The object space approach was originally developed for vector graphics systems. The image space approach was first possible for raster devices, taking advantage of the presence of the frame buffer.
Like stated earlier there are several advantages and disadvantages to the two approaches. In the next sections we will first take a closer look at image space algorithms. Secondly we will say some more about the object space algorithms. Then the combined image space and object space algorithms will be presented. Finally, we will draw some conclusions about these approaches.

2. IMAGE SPACE

Raster displays refresh their screens by reading out the values stored in the framebuffer. Raster displays have become the most common kind of displays in recent years. Mainly because they are the most flexible output devices available. The biggest advantage of the image space approach is that this approach lies very close to the data structure of the frame buffer. Because of this, image space algorithms are often simple and efficient to implement in soft or hardware for raster displays [Fournier et al., 88].

2.1. The Z-buffer algorithm

The most well known of the image space algorithms is the Z-buffer algorithm [Catmull, 74]. For this algorithm another buffer (the Z-buffer) besides the framebuffer is needed with the same number of entries as the frame buffer. The Z-buffer contains the values of the z-coordinates of the nearest objects in the scene.
Suppose a three dimensional (or 3D) image consists of 3D polygons. Then the algorithm pro-

ceeds as follows:
The Z-buffer is initialised to zero, representing the background plane. The frame buffer is initialized with the colour of the background plane. Now every polygon is scan converted. When the value of the z coordinate of a point in the polygon is nearer to the viewer than the z_value already stored in the Z-buffer, then the Z-buffer is updated at this point with the value of the z-coordinate of the point and the frame buffer is updated with the colour of that point.
A huge disadvantage of this algorithm are the aliasing effects occurring. Another problem is that the rendering of transparent objects is done incorrectly.

2.2. The A-buffer algorithm

In [Carpenter, 84] an algorithm is described in which these problems are addressed. This algorithm is called the A-buffer algorithm.
Suppose we consider pixels to be square non overlapping tiles covering the screen. The algorithm starts with clipping each polygon against such a square pixel. These pixel polygons are called fragments. A 4x8 bit mask represents the fragments. When all polygons intersecting a pixel have been processed, the bitmasks of the fragments are used to calculate, with boolean operations, the covering of a pixel. With the coverage of a pixel its intensity is calculated.
The advantage of using the bitmasks is that simple boolean operations can be performed upon them in order to do the necessary calculations. A disadvantage of this algorithm is that it saves only a small amount of information for the fragments, which causes inaccuracies. Another limitation of the algorithm is the size of the bitmasks.

Except for the antialiasing problems there are more general problems for image space algorithms. The biggest problem is that all information of the objects is lost. An example of this problem is zooming. When you zoom in on an image, the values in the framebuffer change, although nothing changes for the object in the image itself. Since the values in the framebuffer change, all the calculations have to be repeated in the image space approach.

We have pointed out an image space hidden surface algorithm, the Z-buffer algorithm, and its descendant, the A-buffer algorithm. We have also shown a few advantages and disadvantages of the image space approach. In the next section examples of object space algorithms will be given.

3. OBJECT SPACE

The most important advantage of object space algorithms is that relevant information is maintained. Because of this, it is better possible to interact with the objects and it makes it possible to apply incremental algorithms. Also the colours of the visible polygons have to be calculated only once. In an image space algorithm it is possible that colours are calculated which later will be over written.
Because of these advantages a lot of research is being done in object space. There are even object space algorithms being implemented in hardware.
Object space algorithms can also be used for shadow generation. Shadow generation is very important, because it improves the depth perception of the image.
In the next subsections examples of object space algorithms will be pointed out and an example for shadow generation will be discussed

3.1. Object space algorithms

Object space algorithms iterate over the objects and/or part of objects. In an object space algorithm the output will consist of the visible objects or the visible parts of objects.

3.1.1. Parallel processing

There are several examples in the literature of object space algorithms. One example is described in [Kuijk et al., 88]. In this article a hidden surface removal algorithm is introduced which is specially developed for a new architecture for raster graphics. This architecture is described in [Akman et al., 88]. In this algorithm surface elements are described in a data structure called domain. The hidden surface removal is done by incrementally adding domains to the so called Low Level Display File (LDF). When a new domain is added, the first thing which is checked is, whether the bounding box of the added domain overlaps any of the bounding boxes of the domains already present. If that is so, then it is checked whether the domains themselves overlap. If they overlap, then the part which is overlapped is removed from the domains and is itself added as a new domain. The appearance of such an overlapping section is determined by the domain which was on top. When all the domains are added to the LDF, the domains which are present will all be disjunct. The data structures and the algorithm used, are designed for parallel processing.
Another example of an object space algorithm which works in parallel is described in [Franklin et al., 90]. Here the uniform grid technique is used to obtain parallelism. This technique is described in [Franklin, 80]. Roughly the uniform grid technique works as follows:
First a grid is overlaid on the scene. The fineness of this grid depends on the number and size of the faces present in the scene. Secondly the edges and faces are sorted into the grid cells, were they are intersected to obtain the visible segments in each cell. The visible regions in a cell are calculated by combining the visible segments. Lastly a point inside a visible region is used to find the shading value for that region.

3.1.1. Output sensitive object space algorithms

When you imagine a scene with one large polygon covering a bunch of smaller overlapping polygons, you don't want the hidden surface removal algorithm to calculate the visibility of all the small polygons, but instead you would like the visibility of the large polygon calculated at once. This is what an output sensitive algorithm should take care of. Output sensitive means that the time needed to do the hidden surface removal does not depend only on the number of polygons in the input, but also on the number of polygons in the output. Most of the object space hidden surface algorithms are not output sensitive. However, output sensitive hidden surface removal algorithms have been developed. One such algorithm is introduced in [Overmars et al., 89]. In this paper a basic output sensitive algorithm and two optimizations of that basic algorithm are discussed.
The basic algorithm calculates a visibility map for a set of 3D objects. This is done as follows: The algorithm assumes that the objects can be ordered by their closeness to the viewing point. Suppose we have a set of triangles in 3D space. Each triangle lies in a plane $z = i$. With i being different for each triangle. The visibility map M is constructed by dividing the xy-plane into maximal regions which have the following properties:

- In each region there exists one unique triangle from the set of triangles, or no triangle at all.
- For all points (x, y) in a region R the following holds: The lowest triangle lying above (x, y) is the triangle of the region, or no triangle at all is lying above (x, y).

The planar map resulting from the division described above is then the visibility map.
The basic output sensitive algorithm for the construction of visibility maps described in the paper, first makes a partial visibility map for a group of triangles from the set of triangles. Then another partial visibility map is constructed with another group of triangles from the set of triangles. The size of each next group of triangles depends on the size of the partial visibility map constructed so far. When there are two partial visibility maps constructed, the two get merged together to form a bigger partial visibility map. This continues until there are no more triangles left in the set of triangles.
A more efficient algorithm is obtained by introducing ray shooting along the edges of the visibility map to construct a connected component. Another optimization is obtained by using triangular regions or base triangles instead of the maximal regions discussed before.

3.1.2. Shadow generation

Shadow generation in an image can be done by using hidden surface removal algorithms. In that case the light source is considered as a viewpoint. It is a big advantage for shadow generation if the hidden surface removal algorithm has the same output form as the input form. This is because the shadow polygons which are generated by means of the hidden surface removal algorithm can then be used as input for the actual hidden surface removal. In [Atherton et al., 78] a shadow generation algorithm is described which uses an object space hidden surface removal algorithm, described in [Weiler et al., 77] of which the form of the input is the same as the form of the output.
This algorithm starts with a rough depth sort, such that the nearest polygons to the viewpoint are in the beginning of the list. The first polygon of this list is used to clip the rest of the list into new lists of polygons. One list consists of polygons which lay inside the clip polygon and one list contains the polygons which are outside the clip polygon. The polygons which are located behind the clip polygon are removed since they are hidden from the observer. From the polygons which remain visible after clipping a new current clip polygon is chosen and the whole process is repeated. If a polygon is found which lays in front of the current clip polygon, then that polygon becomes the new clip polygon and the clipping process is repeated.

3.2. Object space algorithms in hardware

There is more and more interest in applying VLSI techniques to graphics systems. All the above mentioned object space algorithms are implemented in software. Although object space algorithms in general are difficult to implement in hardware, there are object space algorithms implemented in hardware. [Abram et al., 86] is an excellent paper in which several approaches in hardware are laid out. The paper covers image space algorithms as well as object space algorithms. All the examples of object space approaches mentioned, consider the subdivision of objects. These subdivisions are done in order to make parallel processing possible.

4. COMBINATIONS OF OBJECT SPACE AND IMAGE SPACE

We have shown a few examples of object space algorithms.However a lot of research has been dedicated to combinations of object space and image space algorithms. In this section we will look at examples of such combined algorithms.

Priority list algorithms can be classified as being a combination of the object space and image space approach. First the polygons of an object are ordered on their depth. Then these polygons are rendered in that order. The ordering of the polygons is done in object space, the rendering is done in image space. Two well known examples of priority list algorithms are the Depth Sort Algorithm and the Binary Space Partition Tree algorithm. In the next two sub-sections we will look at them in more detail.

4.1. The Depth Sort Algorithm (or Painters Algorithm)

The depth sort algorithm was introduced in [Newell et al., 72]. The basic idea behind this algorithm is that when you sort the polygons in order of decreasing depth, then you can draw the polygons on the screen in such a way that the polygons which are closest to the viewer are drawn last. The first step is therefore the sorting of the polygons. Of course we may find polygons which overlap each other cyclically. In this case at least one polygon has to be split to obtain a correct sorted list. The sorting and the splitting of the polygons are done in object space. The second phase in which the polygons are scan converted in back-to-front manner is done in image space.
In the first phase, which is in object space, all the information about the polygons is still available. The second phase, which just draws the sorted polygons on the screen, is in image space and therefore very fast. Thus this algorithm takes advantages of both worlds. A major drawback is that also disadvantages from both approaches are present. The ordering is slow compared to full image space algorithms and in the second phase the information of the visible parts of the polygons is still lost.
The painters algorithm is sometimes used in flight simulators. An example of such a flight simulator system using this algorithm is described in [Schachter, 81].

4.2. Binary Space Partition Trees

A field in which a lot of research is going on involves the binary space partitioning trees (or BSP-trees). This approach was first introduced in [Fuchs et al., 80]. It consists of two phases. In the first phase a binary space partition tree is created, in the second phase the tree is traversed from a specific viewpoint. The tree is created as follows:
Suppose we have a bunch of planar polygons which represent an object. The creation of the tree starts by choosing one of these polygons as the root of the tree. The plane through the root-polygon splits the space into two sub-spaces. One space before the polygon, relative to its surface normal, and one space behind the polygon. All the remaining polygons are divided over these two sub-spaces. If the plane cuts through a polygon, the polygon is split and each of the two parts is assigned to its proper sub-space. This procedure is repeated for the two sub-spaces until every leaf of the tree contains one polygon.
A major drawback of the algorithm is the fact that the number of polygons in the tree can grow very fast. This is due to the splitting of polygons if the plane of the root polygon cuts through

them. There are ways to decrease the number of splitted polygons. One way is to choose the root polygon carefully. The best polygon to choose as root is the one of which the plane cuts through the smallest number of polygons. Finding such a polygon will take a lot of time. A good approximation is to test with just a few polygons and pick the best of them.
In the second phase of the algorithm the tree is traversed to generate a visible surface image. When the viewpoint is known the tree is traversed in such a way that the polygons in the half-space in which the viewpoint lies are displayed last. In this way the polygons closest to the viewpoint are displayed last and overwrite polygons which are farther away.
It can easily be seen that although the first phase of the algorithm is very time consuming, the second part is just a traversal of the tree. When the BSP-tree has been made, an almost real time frame rate can be achieved, without special graphics hardware. When the viewpoint changes the BSP-tree does not have to be recreated. So this algorithm is particulary suited for scenes with static objects in which the viewpoint changes. Unfortunately, when you deal with moving objects the tree has to be rebuilt every time an object moves. This is why the original algorithm is not suitable for scenes with moving objects.

4.2.1. Moving objects in BSP-trees

In the last few years a lot of progress has been made considering the incorporation of moving objects in BSP-trees. One example is described in [Torres, 90]. Here a BSP-tree is a new six-level structure. Each object has its own single tree. This tree is built by making use of halving planes, which ensure that the constructed tree is balanced. This is done because in this case a balanced tree is built faster than an unbalanced one. The different single trees are put together into one tree structure with the use of different types of auxiliary planes. These planes try to divide the scene in such a way, that the objects are not divided by these planes. In some scenes this is not possible and objects get divided or 'sacrificed'. The trees of the objects which stay intact are called saved single trees. After construction, the structure contains four families of auxiliary planes. The saved single BSP-trees and the polygons of the sacrificed objects are inserted in the leaves. The advantage of such a structure is that modifications on the tree can now sometimes be done locally. The use of halving planes for the building of the single trees shows improvement in time too.
Another example of the handling of dynamic scenes is described in [Chrysanthou et al., 92]. In the algorithm described, the emphasis is on deleting polygons from a BSP-tree. The dynamic computation of shadows from a BSP-tree is looked upon too. This last topic is also thoroughly discussed in [Chin et al., 89].

4.2.2. Other usages of BSP-trees

BSP-trees have also been used for the representation of objects consisting of polygons and to perform set operations on them [Thibault et al., 87][Naylor et al., 90].

Some work has been done on the display of BSP-trees. Originally a BSP-tree is displayed from back-to-front depending on some viewpoint. This way the polygons which are closest are drawn over the ones which are further away. In [Gordon et al., 91] a technique is described for displaying BSP-trees front-to-back. Although there is some additional overhead necessary for testing if pixels are already lit, for a large number of polygons this technique is faster than the back-to-front technique.

From the BSP-trees other data structures have emerged. In [Vanecek, 91] a multidimensional space partitioning tree is presented. The purpose of this MSP-tree is the fast classification of points and lines.
Although the building of the BSP-tree is time consuming, the traversal is very fast. With the ongoing research on BSP-trees the building of the tree will become faster as well as the traversal. BSP-trees can, as stated above, be used for classification of points or lines.
The question remains if BSP-trees can ever be used for moving objects. Although the algorithms are improving in speed they are still often to slow for dealing with moving objects. Besides the inefficiency of the building of a BSP-tree, the disadvantages mentioned for the depth sort algorithm are true here also. In the second phase of the BSP-tree algorithm the information about the objects is lost.

5. CONCLUSION

In the past the emphasis has been put on image space algorithms. Especially because these algorithms are very efficient and simple to implement. However the image space approach has also major drawbacks. These drawbacks are the appearance of aliasing effects and the loss of information. Although there are antialiasing techniques defined for post processing an image it can still be argued that even the antialiasing should be done in object space. Or to quote A.R. Forrest in [Forrest, 85]: "Antialiasing must take into account the items being drawn and work in geometric object space rather than in an image space devoid of knowledge of the objects being rendered."
Since object space algorithms are becoming more and more efficient, plus the fact that the number of possibilities with these algorithms increases, the expectation is that in the future more use will be made of object space algorithms. Especially because the information considering the objects, can be reused.

REFERENCES

[Abram et al., 86] Abram, G.D. and H. Fuchs, "*VLSI-Architectures for Computer Graphics*", Advances in Computer Graphics I (proc. Eurographics '86), Pages 189-204.

[Akman et al., 88] Akman, V., P.J.W. ten Hagen and A.A.M. Kuijk, "*A vector-like architecture for raster graphics*", Report CS-R8802, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[Atherton et al., 78] Atherton, P., K. Weiler and D. Greenberg, "*Polygon shadow Generation*", SIGGRAPH 78, Vol. 12, Pages 275-281.

[Catmull, 74] Catmull, E., "*A Subdivision Algorithm for Computer Display of Curved Surfaces*", Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt LAke City, UT, December 1974.

[Carpenter, 84] Carpenter, L., "*The A-buffer an Antialiased Hidden Surface Method*", SIGGRAPH 1984, Vol. 18, No. 3, July, Pages 103-108.

[Chin et al., 89] Chin, N. and S. Feiner, "*Near Real-Time Shadow Generation Using BSP Trees*", Computer Graphics (SIGGRAPH '89 Proceedings), Vol. 23, No. 3, Pages 99-106.

[Chrysanthou et al., 92] Chrysanthou, Y., and M. Slater, "*Computing Dynamic Changes to BSP Trees*", Computer Graphics forum, Vol II, No. 3, Conference issue Cambridge, UK September 1992, Pages C-321 - C-332.

[Foley et al., 90] Foley, J.D., A. van Dam, S.K. Feiner and J. F. Hughes. *"Computer Graphics: Principles and Practice"*, 2nd edition. Addison-Wesley, 1990.

[Forrest, 85] Forrest, A.R., *"Antialiasing in Practice"*, NATO ASI, Series F, Vol. 17, Pages 113-134.

[Fournier et al., 88] Fournier, A. and D. Fussell, *"On the Power of the Frame Buffer"*, ACM Transactions on Graphics, Vol. 7, No. 2, April 1988, Pages 103-128.

[Franklin, 80] Franklin, W.R., *"A Linear Time Exact Hidden Surface Algorithm"*, Computer graphics, Vol. 14, No. 3, 1980, Pages 117-123.

[Franklin et al., 90] Franklin, W.R. and M.S. Kankanhalli, *"Parallel Object-Space Hidden Surface Removal"*, Computer Graphics (SIGGRAPH '90 Proceedings), Vol. 24, No. 4, August 1990, Pages 87-94.

[Fuchs et al., 80] Fuchs, H., Z.M. Kedek and B.F. Naylor, *"On Visible Surface Generation by A Priori Tree Structures"*, SIGGRAPH 80, Pages 124-133.

[Gordon et al., 91] Gordon, D. and S. Chen, *"Front-to-back display of BSP trees"*, IEEE Computer Graphics and Applications, Vol. 11, No. 5, September 1991, Pages 79-85.

[Kuijk et al., 88] Kuijk, A.A.M., P.J.W. ten Hagen, V. Akman, *"An exact incremental hidden surface removal algorithm"*, Advances in Graphics hardware II, Springer-Verlag, EurographicSeminars, 1988, Pages 21-38.

[Naylor et al., 90] Naylor, B., J. Amanatides and W. Thibault, *"Merging BSP Trees Yields Polyhedral Set Operations"*, Computer Graphics (SIGGRAPH '90 Proceedings), Vol. 24, No. 4, Augustus 1990, Pages 115-124.

[Newell et al., 72] Newell, M.E., R.G. Newell, T.L. Sancha, *"A Solution to the Hidden Surface Problem"*, Proceedings of the ACM Annual Conference, Boston, August 1972, Pages 443-450.

[Overmars et al., 89] Overmars M. and M Sharir, *"Output-Sensitive Hidden Surface Removal"*, 30th Annual Symposium on Foundations of Computer Science, 30 Oct. - 1 Nov. 1989, Pages 598-603.

[Schachter, 81] Schachter, B.J., *"Computer Image Generation for Flight Simulation"*, IEEE Comput. Graphics and Appl., Vol. 1, October 1981, Pages 26-68.

[Sutherland et al., 74] Sutherland, I.E., R.F. Sproull and R.A. Schumacher, *"A characterization of Ten Hidden-Surface Algorithms"*, ACM Computing Surveys, Vol. 6, No. 1, March 1974, Pages 1-55.

[Thibault et al., 87] Thibault, W.C. and B.F. Naylor, *"Set Operations on Polyhedra Using Binary Space Partitioning Trees"*, SIGGRAPH 87, Pages 153-162.

[Torres, 90] Torres, E., *"Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes"*, Proceedings Eurographics 1990, Elsevier Science Publishers B.V., Pages 507-518.

[Weiler et al., 77] Weiler, K and P. Atherton, *"Hidden surface removal using polygon area sorting"*, SIGGRAPH 77, Vol. 11, No. 2, 20-22 July 1977, Pages 214-222.

[Vanecek, 91] Vanecek, G.Jr., *"Brep-index: a multidimensional space partitioning tree"*, Internat. J. Comput. Geom. Appl., Vol. 1, No. 3, 1991, Pages 243-261.