A DENSITY BASED APPROACH TO OCCLUSION CULLING

A PROJECT REPORT

submitted by

AASHISH DUDDUKURI B120850CS

in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

under the guidance of

DR. PRIYA CHANDRAN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY CALICUT
NIT CAMPUS P.O, CALICUT
KERALA, INDIA 673601
MAY 2016

ACKNOWLEDGEMENTS

A project's success is never an individual effort, but is the collective endeavor of many a people. It would be gross incompetence on one's part, to not acknowledge such contributions. Firstly I am grateful to **Dr. Priya Chandran** *(Professor, NIT Calicut)* for guiding and supporting through out the project, with the constant advice and constructive criticisms. Her support has been immeasurable and for this I owe to her my sincerest gratitude. I would also like to extend my gratitude to **Ms Anu Mary Chacko, Mr Jimmy Jose** and the other panel members **Ms Febna, Mr Gaurav and Mr Ashwin** for their patience and providing valuable suggestions/feedback to the project. Also a special mention to the whole of Computer Science and Engineering Department fraternity at NIT Calicut for equipping me with the tools and knowledge to handle this project effectively and make it successful.

DECLARATION

I hereby declare that this submission is my/our own work and that, to the best of my/our knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text".

Place: Kozhikode
Date: 12th May 2016

Signature:

Name: Aashish Duddukuri
Roll No.: B120850CS

CERTIFICATE

*This is to certify that the project report entitled: "***A DENSITY BASED APPROACH TO OCCLUSION CULLING** *" submitted by Sri* **Aashish Duddukuri, B120850CS** *to National Institute of Technology Calicut towards partial fullfilment of the requirements for the award of Degree of Bachelor of Technology in Computer Science Engineering is a bonafide record of the work carried out by him/her/them under my/our supervision and guidance.*

Signed by Dr. Priya Chandran(Project guide)

Place: Kozhikode
Date:12th May 2016

Signature of Dr. Abdul Nazir,
(Head of Department,
Computer Science and Engineering)

Office Seal

Abstract

Interactive virtual environments are seen or used in varied fields of both research and entertainment. While making these worlds impressive and detailed is very appealing, the graphics hardware always limits the performance and the interactivity of such worlds. Gaining performance in such rendering processes has become very important to make the environments sufficiently detailed as well as interactive. There exist many different methods to improve the rendering processes in general, some of them being Level of Detail, Viewfrustrum culling and Backface culling. One of the most important effective way to gain performance is Occlusion Culling. The rendering time of a given polygonal dataset is mainly determined by the number of polygons(objects) sent to the graphics hardware. Rendering acceleration algorithms, like occlusion culling have tried to reduce the set of rendered polygons(objects) sent to the graphic hardware. Occlusion culling literally means culling(removing) occluded(obstructed) primitives from the dataset sent to the graphics hardware. By removing such obstructed primitives we reduce the set of rendered polygons thereby improving(accelerating) the rendering. Since the problem boils down to finding obstructed polygons, these class of algorithms are also called Visibility detection algorithms. This project attempts to to delve into the various algorithms pertaining to occlusion culling. In particular the feasibility of Binary space partitioning algorithms were closely studied. Most of the classic algorithms related to occlusion culling try to sort the polygons(objects) based on various properties like distance from view point etc. Properties like density of the environment are not usually considered. We try to use the trivial relationship of density and distance to occlusion, namely visibility is inversely proportional to density and inversely proportional to distance. We use this information to find out a composite function(referred to as visibility estimate) which is basically an indication of the probability of a polygon being occluded or not.

Contents

List of Figures

Chapter 1

Introduction

The rendering time of a given polygonal dataset is mainly determined by the number of polygons(objects) sent to the graphics hardware. Rendering acceleration algorithms, like occlusion culling have tried to reduce the set of rendered polygons(objects) sent to the graphic hardware. Occlusion culling literally means culling(removing) occluded(obstructed) primitives from the dataset sent to the graphics hardware. By removing such obstructed primitives we reduce the set of rendered polygons thereby improving(accelerating) the rendering process.

1.1 Problem Statement

The problem is 'Given a set of geometric primitives(objects), a view location and a view direction, find all the occluded primitives more efficiently than presently available methods'. This involves the formulation of an analysis about various relative depths of the geometric primitives so as to find the occluded ones.

1.2 Literature Survey

While there are many different approaches to solve this problem, all such algorithms show some commonality, as presented by I. E. Sutherland [1]. The properties include coherence and sorting. Sorting is justified because visibility to a computer is nothing but finding the closest primitive from the given dataset and usually algorithms try to find keys for each polygon based on its various geometric attributes, these keys are then sorted to get to get the best polygon to render first. While approaches like Painter's algorithm[3] use a very trivial key, namely the Z coordinate, others use a key compounding various geometric attributes into one. Almost all algorithms use some sort of coherence so as to reduce proportions of work involved in sorting. The word coherence comes from Latin which means "to stick together", in a general sense being coherent basically means being similar or in unison. In most environments there is always some "similarity", be it between neighboring polygons or the ones in the corners. Such coherences are called spatial coherences these are basically similarities in spatial domain/ in 3D space. While trying to use spatial relationships is one side of the coin, many methods use what is called as Temporal coherence. This pertains to time related similarities i.e. using the status of the the previous state to improve the sorting. In most interactive environments the view point mostly changes only by a little per second so most polygons retain same properties as before, such information can be used to have drastic improvements in performance. Almost all Occlusion culling algorithms have these properties but are used with out actually realizing it. In the following sections various basic occlusion culling techniques are described and analyzed.

1.2.1 Painter's Algorithm[3]

This is the most intuitive as well as trivial algorithm. The basic concept being, all polygons in the dataset are first sorted according to their Z coordinate. This is nothing but the perpendicular distance of a polygon from the specified view point. The polygons are then rendered from farthest to closest. While this sounds trivial it suffers from many problems, the important one being that polygons usually don't have a constant Z coordinate. This causes problems in rendering overlapping or intersecting polygons. Apart from this there is a major misutilisation of GPU due to rendering every polygon.

1.2.2 Z-Buffer Algorithm[1]

This is again one of the most frequently used hardware rendering technique in practice. The algorithm maintains an extra data structure corresponding to every pixel, which is called the Z Buffer. This is also called the depth buffer due to the fact that it stores the depth corresponding to each coloured pixel. When ever a new polygon is rendered, each pixel's Z coordinate(corresponding to the polygons projection) is compared with the respective depth value in the Z buffer. Only when the present polygon's pixel is closer than the respective depth value in the buffer, is it updated to the buffer. Since this algorithm deals with many low level aspects of the GPU it is the favoured hardware rendering technique.

1.2.3 Warnok's Algorithm[2]

This is an elegant Divide and Conquer technique. It, like all divide and conquer algorithms, classifies polygons present in the current division into trivial and non trivial cases. While the trivial ones are easily handled, non trivial cases are again subdivided. Here also spatial coherence is extensively used to figure out whether a subdivision corresponds to a trivial or non trivial case. The algorithm uses the concept of surrounding polygons namely, if a polygon is both surrounding and closest to the view point then that subdivision may be rendered with that surrounding polygon.

1.2.4 Binary Space Partitioning[4]

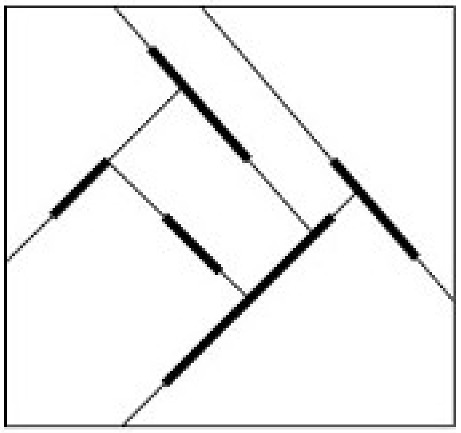
Binary space partitioning is basically a method to establish various spatial relationships between the polygons of the given dataset. It uses a separate tree called as the Binary space tree to help store these spatial relationships between the various polygons. Most BSP algorithms use the usual back to front approach which causes wastage of resources. The space is partitioned along a hyper plane into two half-spaces, then either half-space is partitioned recursively until every subproblem contains only a trivial fraction of the input object. The drawbacks of such a system and a much more efficient implementation is described by Gordon D[4]. It utilizes a front to back approach to display BSP trees which improves upon the traditional BSP algorithms. Gordon combines another traditional algorithm called scanline algorithms with the BSP to achieve improved performance. Traditional scanline algorithms have a data structure known as "dynamic screen" which is nothing but another representation of the n x n screen as a n sized linked list. This dynamic screen combined with the active edge tables of each primitive and utilizing the relationships established by the BSP and using a merge like process Gordon improves upon the traditional BSP algorithms.

Though BSP is meant to be a preprocessing step, for environments with large no of primitives this consumes a lot of resources. Partitioning only a part of the the given set of polygons requires constant updating of the BSP trees, which would be costly. These problems and their solutions are described in Chrysanthou et al[5]. Chrysanthou describes three criteria necessary for to support dynamic changes in 3D scenes:

- ability to alter the view location and direction
- ability to add a polygon to into the dataset and the tree
- ability to delete a polygon from the dataset and the tree

Both 1 and 2 can trivially be handled by BSP trees; altering view direction basically is traversing the tree in a different order. Adding a new polygon is also trivial as it just means recursively partitioning the primitive into the BSP tree. The third criterion is much more difficult; deleting a primitive from a scene is hard in a BSP tree because a primitive may be split into the front and back leafs of the node, and therefore might require the change of the whole BSP tree which is computationally not feasible.

Figure 1.1: An Auto-partition



Since Binary space partitioning involves splitting of the object space, it usually ends up framnenting the polygons also. BSP is a preprocessing step due to which the BSP tree always needs to be in the memory, this results in the bottle neck of the space complexity being the size of the BSP tree. Ideally, for a perfect bsp tree we would like the size of the tree to be the number of polygons. This happens in the case where all hyperplanes chosen were the input polygons and when there is no fragmenting of polygons taken place and such partitions are called Auto-partitions. Auto-partitions have been used extensively because of their ease in implementation. In the same lines a perfect BSP refers to the partitioning where no input primitives are fragmented. It is also trivial that the time complexity of the algorithm depends on the height of the BSP. It is also interesting to note that attaining minimum average height will not always result in minimum fragmenting of the polygons.

Minimizing such fragmenting of polygons is clearly desirable. Unfortunately, efforts finding an efficient algorithm for computing optimal binary partitions turned out to be idle: Mark de Berg[7] proves that computing optimal partitions is an NP-hard problem. In fact, it is even NP-hard to decide whether a set of segments admits a perfect partition.

Their hardness proof is based on a new 3-SAT variant, monotone planar 3-SAT. They derive and define from a special version of the satisfiability problem, which they later go on to prove is NP-complete. The problem of finding optimal auto-partitions is then fit into this variant thereby proving that this(finding optimal auto-partitions) in fact is NP-hard.

Chapter 2

Design

2.1 Motivation

It is logical to see that density of a scene contributes in whether a polygon is occluded or not i.e. areas in the environment with a much higher density tend to have their polygons occluded more frequently than areas with lower polygon density. Similarly we can see that distance of the polygon from view point also contributes in finding whether the respective polygon is occluded. Farther polygons clearly have an increased probability of being occluded, where as closer ones have less probability of being occluded. The relationships are as follows:

Instead of considering these relationships separately we try to compound them into a single estimate. This will make sure that ones with lower density and closer to the view point are rendered first and before the polygons which are farther away from the view point and with higher density. We call this compound value for each polygon a visibility estimate

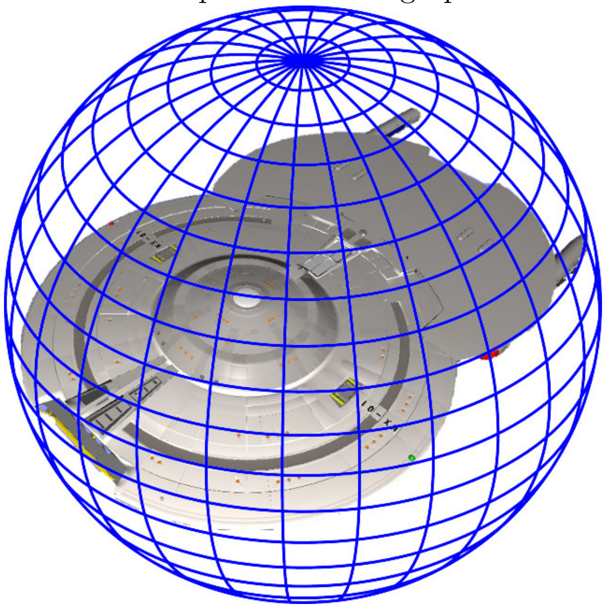
$$estimate = c.density^{x}.depth^{y} \tag{2.1}$$

Here the values of x and y can be changed, based on the importance given to either attribute. If density was given more preference x¿y and similarly if depth is given more preference then y¿x. This gives us the flexibility to experiment with different types of these estimate functions. While this sounds very logical, it is riddled with problems. The first one being the fact that computing density is extremely costly, it costs as much as rendering the whole scene with brute force. Depth also suffers with same problems as mentioned in the Painter's Algorithm[3] namely, not all polygons can have a constant depth/ Z coordinate value. These reasons make the finding the suggested estimate computationally intensive and impractical to use. But we don't usually need the exact estimate, getting an approximate value of the estimate might help reduce the complexity and make it computationally feasible.

2.2 Design

To make this technique viable we use various Bounding algorithms. As the name suggests these algorithms are dedicated to find various geometric shapes which completely bind/surround the object in question. They convert complex and detailed objects into trivial geometric shapes. These trivial geometric shapes can be easily be analyzed to get density. Further more, by using this we also get an estimate of the Z coordinate/ depth which basically is distance from centre of this primitive geometric shape to the view point. There are two different kinds of Bounding algorithms, with one being Bounding box and other being Bounding sphere algorithms. As the names suggest the first one tries to find a box /cuboid which encloses the object completely and the other one tries to find the a sphere enclosing the while object in question. Though bounding box algorithms can be used, we will be referring to only bounding sphere algorithms for the rest of the paper. It is important to note that finding just Bounding spheres in not enough, but finding the minimum bounding sphere is our main goal.

Figure 2.1: An example of Bounding Sphere of an object



Algorithms to find the minimum bounding spheres are extensively studied and largely have already been figured out. There are various fast and simple bounding sphere construction algorithms with a high practical value in real-time computer graphics applications, which find an approximately minimum bounding algorithm. For any object,however detailed and complex, there are already methods in place to find bounding sphere in linear time. Due to this reason we will be cease to over analyze these algorithms and move ahead.

Now that we have proven that the suggested technique is viable in this case. We will try to define properly what this density actually means. This density pertains to each and every polygon, it is the number of polygons whose projection intersects with the projection of the polygon whose density we try to find. We realize this by having another data structure pertaining to each pixel called Counter nxn which is initialized to zero. We then project all the bounding spheres onto the screen and for every projection we increase the count of the corresponding pixels in the counter data structure. This data structure basically contains information of the no of overlapping bounding polygons per pixel. From this data structure we get the no of overlapping polygons for a certain view point for each polygon, which basically is the so called density per polygon we were trying to find.

$$Density(p) = No.\ of\ polygons\ overlapping\ with\ p \tag{2.2}$$

2.3 Related Work

El-Sana[8] uses a similar concept but instead of finding density with respect to each polygon, they calculate a ”solidity” value pertaining to each cell. They divide the whole 3D environment into cells of various sizes, with each cell pertaining to a cubical region in the environment. This solidity is calculated by utilizing the concept of Face projection. In this technique they project polygons in the cell onto the 6 sides of the cell, which they refer to as projection area of the cell. Solidity then computed as the ratio of no of rays passing through the 6 sides to total area of the cell. This value is then assumed to be the indication of visibility which is used to sort and render cells having values of less solidity to more.

Chapter 3

Implementation

The algorithm was implemented in C and C++ on Linux. The powerplant model, which is extensively used in various fields of graphics to undertake tests was used mainly to test the implementation. The experiment was carried upon an Intel i5 system with 512MB of dedicated GPU. For testing usually, the view point was either fixed and rotating or was moving linearly in straight lines.

```
Input: Set of polygons $[N]$
Output: Rendered image
begin
    forall $i \in [N]$ do
        $bound[\,] = MinBoundSphere(i)$// Min Bound sphere is found
    end
    density[ ] = CalcDensity([$N$],$bound$)// find densities of each
        of the polygons
    estimate[ ] = CalcEstimate([$N$], $density$,bound) // find estimates
        of each of the polygons using function
    sort([$N$], $estimate$)
    forall $i \in [N]$ do
        $render(i)$// final rendering
    end
end
```

Algorithm 1: Density Based occlusion Culling

Another variant on this algorithm which is a specified budget based algorithm was also implemented. Doing such budget specific computation gives appalling visuals but the frame rate is very high when compared to the Z buffer. The visuals of such an implementation usually are ridden with various ”Black patches” which are basically areas where computation was prematurely terminated. Such algorithms usually help these highly detailed interactive environments to be rendered at acceptable frame rates even for Computers with very low GPU capacity.

```
Input: Set of polygons [$N$], Budget
// Here budget is max no of polygons to be analysed
Output: Rendered image
1  begin
2      forall $i \in [N]$ do
3          $bound[\,] = MinBoundSphere(i)$// Min Bound sphere is found
4      end
5      density[ ] = CalcDensity([$N$],
        $bound$)// find densities of each of the polygons
6      estimate[ ] = CalcEstimate([$N$], $density$,bound) // find estimates
          of each of the polygons using function
7      sort([$N$], $estimate$)
8      forall $i \in [N]$ do

9          if $i < Limit$ then
10             $render(i)$// final rendering
11         end
12         else
13             break
14         end
15     end
16 end
```

Algorithm 2: Density Based occlusion Culling with a budget

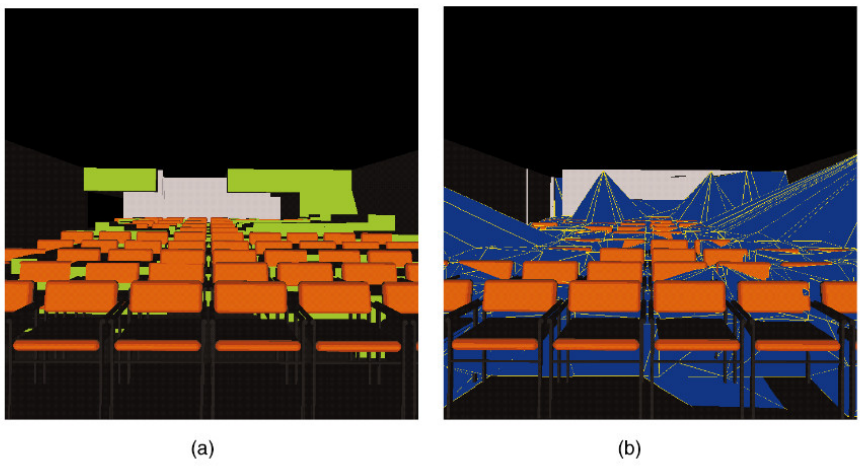
In the next chapter we try to see how our proposed algorithm stacks up with respect to the other primitive algorithms, by running various tests on the decided datasets. The implementation uses various different libraries like SMFL(Simple and Fast Multimedia Library) to help reduce code complexity.

Chapter 4

Results

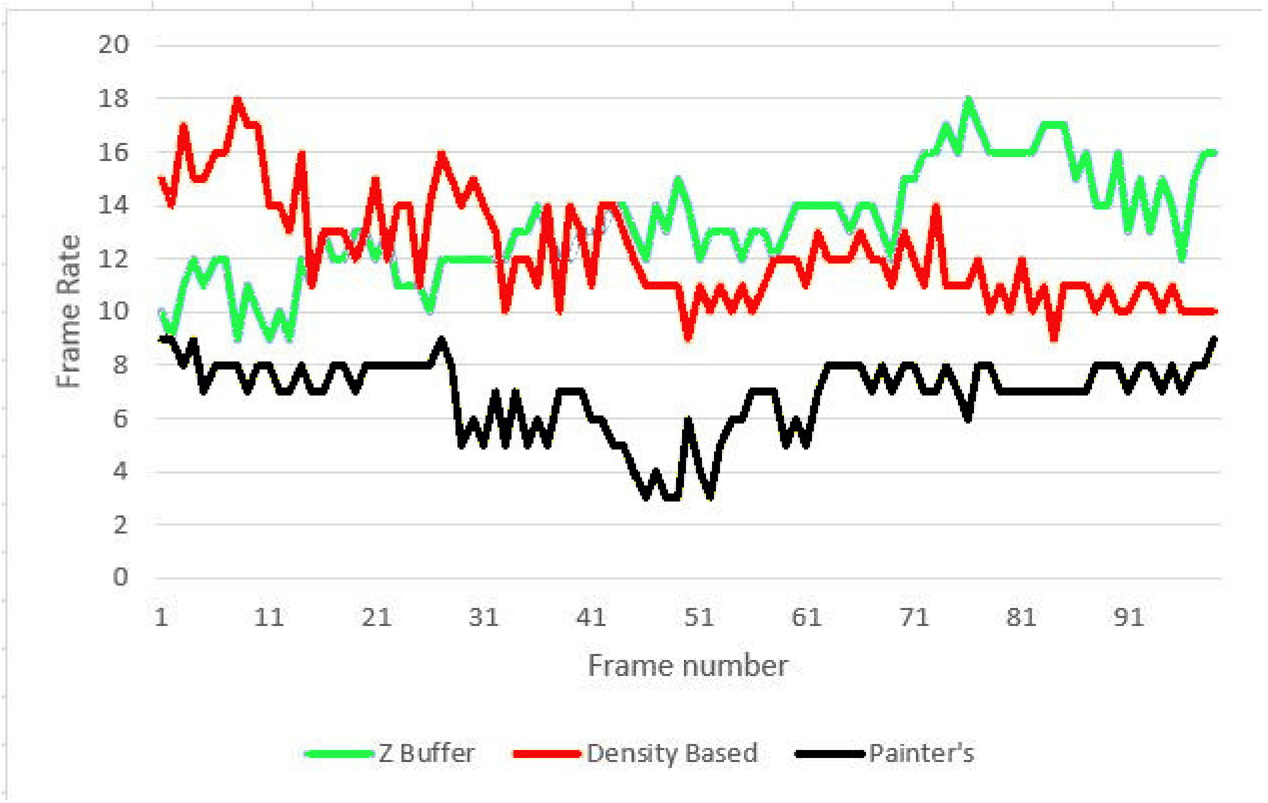
The first set of figures clearly potrays how the black patches show up when we use the algorithm involving a budget. It is interesting to note that systems which lack GPU capability also show similar outputs, as they are forced to stop the algorithm because of lack of GPU power. While (a) shows how a full run of the algorithm looks like, we can see the incomplete budgeting in (b)

Figure 4.1: An example of budgeted version of the algorithm



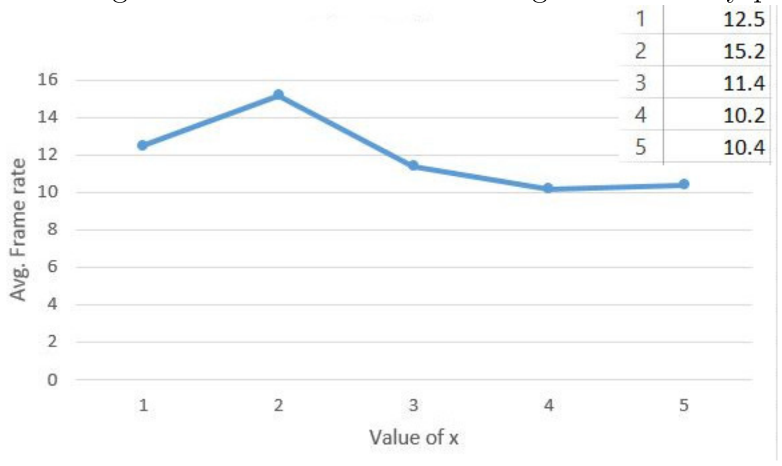
The second set of figures are a clear indication of the performance of the algorithm, we make a comparison with Z buffer algorithm, painters algorithm and our Density based algorithm. While the X axis corresponds to the frame rate, Y axis corresponds to frame number or is basically an indication of time.

Figure 4.2: Comparison with Z buffer and Painter's



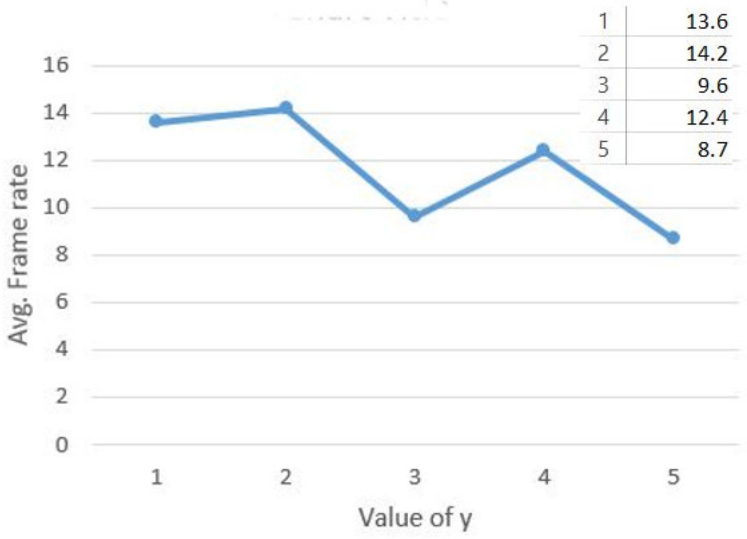
In the third set of figures we try to use various estimate functions and compare them to find out whether there is improvement in the algorithm's performance with more preference to either density or depth. Here while Y axis is the average frame rate, X axis corresponds to the differant values for x and y in the estimate function *(2.1)*

Figure 4.3: Average frame times with various degrees of density preference



1	12.5
2	15.2
3	11.4
4	10.2
5	10.4

Figure 4.4: Average frame times with various degrees of depth preference



1	13.6
2	14.2
3	9.6
4	12.4
5	8.7

4.1 Remarks and Observations

The first very obvious observation is that our proposed algorithm has better frame rates than Painter's algorithm[3]. It can also be observed that in the beginning, our algorithm clocks better than Z buffer Algorithm[1], but the Z buffer algorithm gradually improves and registers higher frame rates as time passes. While the graph corresponding to density preference seems to have a maximum at 2, the graph corresponding to depth preference is pretty erratic.

We now study all these results and try to find out the reasons for our previous observations, which will help us to find the proposed algorithm's flaws and also contribute in improving it.

Chapter 5

Conclusions

Firstly, among the various estimate functions studied we see that giving density a little preference over depth gives desirable and improved results, so such estimate functions should be considered. When it comes to depth it seems that it doesn't have any consistent effect on performance by increasing its preference in the function. After various runs it was found that the following estimate function perform better than others.

$$estimate = c.density^2.depth^1 \tag{5.1}$$

Secondly, we see that out algorithm is almost always better than painters algorithm. But the algorithm doesn't compare well with Z buffer algorithm, the reason being that Z buffer Algorithm is very robust as it utilizes many low level concepts. We feel that the algorithm is done justice only by comparing the algorithm with other object space algorithms and not image space algorithms.

Our algorithm deals with instances, where there is varied density across the scene, very well. This is because large parts of the scene with minimal density will be dispatched off first, while the major resources are used to deal with areas of high density. In such cases our algorithm is better than other primitive algorithms.

We see that Z buffer algorithm starts off pretty weak and gains traction as time passes, where as our algorithm seems to have good frame rates in the beginning and looses steam as time passes. This may be due to the reason that our test starts from outside the powerplant to inside. When the view point is outside there is varied density in the scene, namely the sky(low density) power plant itself(high density). Therefore it runs better in such a case, but as we move into the powerplant the density becomes high and constant due to which our algorithm doesn't replicate its prior performance and our frame rates drop off.

But on average our algorithm doesn't do as well as the Z buffer Algorithm. While our Algorithm clock in 12-14fps, Z buffer clocks in on 15-20fps in some cases reaching as high as 28fps.

5.1 Problems

The biggest problem with this algorithm is the problem relating to partial visibility in case of budgeting or lack of GPU. Though it happens only in some cases it is anti immersive and surely need to be fixed.

As we can see when we take environments with minimal/constant density, then our algorithm ends up doing substantially lot of work that primitive algorithms. Due to which it doesn't compete well with other algorithms in such cases. But such cases are only theoretical and almost never occur in practical purposes.

The final rendering done by the GPU, represented as render() in pseudocode seems to work on all polygons, but the implementation adds a hardware check pixelEmpty(). This check stops render when there are no more pixels to be coloured.

The other major problem with the algorithm is that it involves finding bounding boxes/spheres twice. Our algorithm gives a stream of polygons with increasing probability of occluding, other than that we don't specify any info about either partial or full occlusion. This check for partial/ full occlusion has to be done again during the final rendering for the correctness. This check utilizes queries which again use bounding boxes/spheres of a given polygon. Removing this double work proved unsuccessful as, render() is a very low level function and editing it proved very challenging.

5.2 Suggested Improvements

One of the major improvements to the present algorithm is to include temporal coherence. We try to utilize the history of the polygons in previous state to try to predict its visibility. We can change the estimate to reflect such a change and this will hopefully improve the algorithm.

render() could be improved to stop the double work of finding bounding polygons. Partial / complete occlusion check can be clubbed with the process of finding densities, so as to help in final rendering process.

As the problem of black patches has already been discussed, we may try to improve this by integrating the concept of Level of Detail. Usually LOD is determined by the distance of the object from view point. We try to decrease an object's complexity and its detail based on how far it is. This concept can be integrated in our present algorithm, so that after budget is finished the rest of the polygons can be rendered in a low level of detail as they most probably are partially occluded. This will help save a lot of GPU power to be utilized in the next frame.

Bibliography

[1] Ivan E. Sutherland , Robert F. Sproull , Robert A. Schumacker, *A Characterization of Ten Hidden-Surface Algorithms*, ACM Computing Surveys (CSUR), v.6 n.1, p.1-55, March 1974.

[2] Warnock J.E, *A Hidden surface Algorithm for Computer Generated Halftone pictures*, Computer Science Department,University of Utah, TR'1-15, (June 1969).

[3] Foley James, Feiner Steven, K. Hughes John F. *Computer Graphics: Principles and Practice.* Reading, MA, USA: Addison-Wesley. p. 1174, 1990.

[4] Gordon, D., Chen, S. *Front-to-Back Display of BSP Trees*, IEEE CG A, 11(5), 79-85 (1991).

[5] Chrysanthou, Y., and Slater, M., *Computing Dynamic Changes to BSP trees*, Computer Graphics Forum (EUROGRAPHICS '92 Proceedings), 11(3), 321-332, sep 1992.

[6] Bruce Naylor, John Amanatidest and William Thibault *Merging BSP Trees Yields Polyhedral Set Operations*, Computer Graphics (SIGGRAPH '90 Proceedings), AT T Bell Laboratories, ACM Press, New York, NY, USA, 1990, pp. 115124

[7] Mark de Berg, Amirali Khosravi *Optimal Binary Space Partitions in the Plane*, IEEE CG A, 11(5), 79-85 (2010).

[8] Jihad El-Sana, Neta Sokolovsky, Claudio T. Silva *Integrating Ocllusion culling with View-Dependent Rendering* , IEEE Computer Society, Washington, DC, USA, VIS 01,371378.