G.KOLDAŞ

METU 2008

EFFICIENT VISIBILITY ESTIMATION FOR DISTRIBUTED VIRTUAL URBAN ENVIRONMENTS

GÜRKAN KOLDAŞ

FEBRUARY 2008

EFFICIENT VISIBILITY ESTIMATION FOR DISTRIBUTED VIRTUAL URBAN ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÜRKAN KOLDAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

FEBRUARY 2008

Approval of the thesis

EFFICIENT VISIBILITY ESTIMATION FOR DISTRIBUTED VIRTUAL URBAN ENVIRONMENTS

submitted by **GÜRKAN KOLDAŞ** in partial fullfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Volkan Atalay _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Veysi İşler _____
Supervisor, **Computer Engineering, METU**

Prof. Dr. Rynson W. H. Lau _____
Co-Supervisor, **Computer Science, University of Durham**

Examining Committee Members:

Prof. Dr. Uğur HALICI _____
Electrical and Electronics Engineering, METU

Assoc. Prof. Dr. Veysi İşler _____
Computer Engineering, METU

Assoc. Prof. Dr. Halit OĞUZTÜZÜN _____
Computer Engineering, METU

Assoc. Prof. Dr. Harun ARTUNER _____
Computer Engineering, Hacettepe University

Assist. Prof. Dr. Tolga CAN _____
Computer Engineering, METU

Date: 06.02.2008

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Gürkan KOLDAŞ

Signature :

ABSTRACT

EFFICIENT VISIBILITY ESTIMATION FOR DISTRIBUTED VIRTUAL URBAN ENVIRONMENTS

Koldaş, Gürkan

Ph.D., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Veysi İsler

Co-Supervisor: Prof. Dr. Rynson W.H.Lau

February 2008, 92 pages

This research focuses on the utilization of occlusion culling for the real-time visualization of distributed virtual urban environments. Today's graphics hardware renders all the primitives in any order and uses z-buffer to determine which primitives are visible on a per-pixel basis. However, visibility is computed in the last stage of rendering pipeline and every rendered primitive is not visible in the final image. Early culling of the invisible primitives in a complex scene is valuable for efficiency in the conventional rendering pipeline. This may reduce the number of primitives that will be processed in the rest of the pipeline. In this thesis, we propose an efficient visibility estimation method for distributed virtual urban environments. The proposed method is based on occlusion culling to identify and cull the occluded parts of the scene. This not only computes conservative potential visible set (PVS) for each client but also assures the computed PVS to be available at the client on-time and reduces the network traffic by grouping the clients which may see each others dynamically.

Keywords: Visibility, Occlusion Culling, Distributed, Area of Interest, Interactive

ÖZ

DAĞITIK SANAL ŞEHİR ORTAMLARI İÇİN ETKİN GÖRÜNÜRLÜK KESTİRİMİ

Koldaş, Gürkan

Doktora, Fen Bilimleri Bölümü

Tez Yöneticisi: Doç. Dr. Veysi İsler

Ortak Tez Yöneticisi: Prof. Dr. Rynson W. H. Lau

Şubat 2008, 92 sayfa

Bu araştırma dağıtık sanal kentsel ortamların gerçek zamanlı görselleştirilmesi için kapatma ayıklaması kullanımı merkezlidir. Günümüz grafik donanımları tüm temel öğeleri her hangi bir sırada gerçeklemekte ve z-arabelleği kullanarak piksel seviyesinde görünebilir olanları belirlemektedir. Ancak, görünürlük görsel gerçekleme ardılış düzeninin en son aşamasında hesaplanmakta ve gerçeklenen her bir temel öğe en son görüntüde görünmemektedir. Karmaşık sahnelerde görünmeyen temel öğelerin erken ayıklanması görsel gerçekleme ardışık düzeninde etkinlik açısından önemlidir. Bu sayede görsel gerçekleme ardışık düzeninin geri kalan kısmında işlenecek olan temel öğe sayısı azaltılabilir. Bu tezde, dağıtık sanal şehir ortamları için etkin bir görünürlük hesaplama yöntemi önerilmektedir. Önerilen yöntem, sahnenin görünmeyen kısımlarının tanımlanması ve ayıklanması amacıyla kapatma ayıklaması temellidir. Bu sayede her bir katılımcı için kararlı Muhtemel Görünürler Kümesi (MGK) hesaplanmakla kalmaz ayrıca hesaplanan MGK'nin zamanında katılımcıda olması garanti

edilir ve dinamik olarak birbirini görebilecek katılımcıların gruplanması ile ağ trafiği azaltılır.

Anahtar Kelimeler: Görünürlük, Kapatma Ayıklama, Dağıtık, İlgi Alanı, Etkileşimli

To My Parents

ACKNOWLEDGMENTS

The author wishes to express his deepest gratitude to his supervisor Assoc. Prof. Dr. Veysi İŞLER and co-supervisor Prof. Dr. Rynson W. H. Lau for their guidance, advice, criticism, encouragements and insight throughout the research.

The author would also like to thank

Dr. Çağatay ÜNDEĞER and Şafak Burak ÇEVİKBAŞ for their suggestions and comments.

My colleagues: Capt. Levent KOCABIYIK, Cdr. Cem Ali DÜNDAR, Lt.Cdr.Dr. Tolga AYAN, Lt.Cdr. Necmi ÖZDEMİR and Lt. Sadettin GÜNEŞ for their supports.

My wife Dilşat KOLDAŞ for her unlimited patients and supports.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AoI : Area of Interest
BSP : Binary Space Partitioning
CPU : Central Processing Unit
CoVP : Center of View Plane
CVE : Collaborative Virtual Environment
CVP : Convex Vertical Prism
DDM : Data Distribution Management
DIS : Distributed Interactive Simulation
DoF : Degree of Freedom
DVE : Distributed Virtual Environment
GPU : Graphical Processing Unit
HLA : High Level Architecture
HOM : Hierarchical Occlusion Map
HSR : Hidden Surface Removal
HZB : Hierarchical Z-Buffer
LoD : Level of Detail
MGK : Muhtemel Görünürler Kümesi
MMOG : Massively Multiplayer Online Games
NVE : Networked Virtual Environment
P2P : Peer-to-Peer
PVS : Potential Visible Set
RTI : HLA Runtime Infrastructure
OH : Occlusion Horizon
VbAoI : Visibility-based Area of Interest
VE : Virtual Environment
ΔH : Delta Horizon or Incremental Occlusion Culling Method

CHAPTER 1

INTRODUCTION

Visibility has broad application areas in computer graphics, architecture, computational geometry, computer vision, robotics, telecommunications, sensors and other research areas. The importance of visibility in computer graphics increases parallel to the recent developments in computer technology and increased demand on interactive simulations and games. Visibility is still an open research area from the beginning of the studies in computer graphics.

Visibility researches in computer graphics focus on estimation of visible primitives in the scene that may have many complex and overlapped objects. A visibility method aims to compute the photo-realistic image by assuring that the primitives behind do not incorrectly occlude the front ones. Current graphics hardware renders all the primitives in any order and utilizes z-buffer to determine which are visible on a per-pixel basis [1]. However, every transmitted primitive to graphics hardware is not visible in the final image and visibility is computed in the last stage of the rendering pipeline [2].

Rendering is a term used for the overall process of going from a database representation of a 3D object to a shaded 2D projection on a view surface [3]. The collection of these processes are usually known as graphics pipeline or rendering pipeline. Although the processes inside the pipeline may be different for different rendering systems, the conventional rendering pipeline for a polygon rasterization based rendering is given in Figure 1.1 [4].

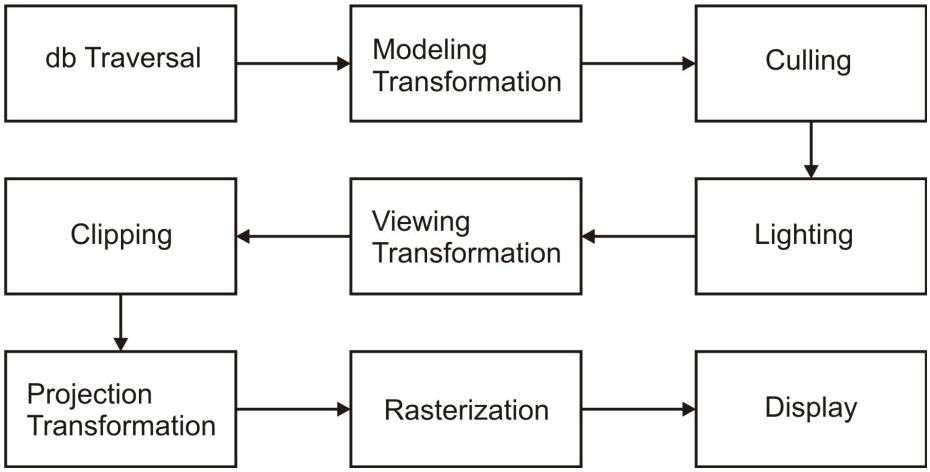


Figure 1.1: Conventional rendering pipeline.

Rendering pipeline performs a collection of processes for each polygon transmitted to the pipeline. At the end of the pipeline, a polygon is either rasterized or discarded. In Figure 1.1, dbTraversal is the initial phase which transmits the polygons to the pipeline for rendering. Modeling transformation transforms the object geometry from local coordinates to the world coordinates. Culling phase discards polygons that can not be visible from the view point. The place of culling in pipeline may change with respect to visibility methods. We assume that view frustum culling is only performed in Figure 1.1. Lighting phase calculates the color of vertices by using shading parameters and an illumination model. [4]. Viewing transformation transforms objects from world coordinates to the viewer's viewing frustum. Clipping is performed for the view frustum in the next phase. Projection transformation transforms 3D object to 2D view plane by projecting its vertices and calculates depth information for vertices. In the rasterization phase, polygons are scan converted. It is the operation of computing the screen coordinates of the projected polygons. The output of the rendering pipeline is the rendered pixels

which are sent directly to frame buffer or to an off-screen buffer to be utilized in the consecutive tests.

Rendering pipeline processes are grouped into three different stages according to their concerns as follows [2]:

1. *Traversal:* The objects in the virtual environment are traversed for rendering. A common method is to select the objects in the viewing frustum.

2. *Transformation:* The geometry of each traversed object is transformed from the world coordinate system to the screen coordinate system.

3. *Rasterization:* The geometry of each object is rasterized and written into the frame buffer after transformation. A fragment for each corresponding pixel in the frame buffer is generated. The term of fragment is used for all of the data needed to generate a pixel in the frame buffer. The depth information of each fragment is compared to the stored depth-value in the z-buffer to test the fragment's visibility.

The visibility test is performed in the rasterization stage of the rendering pipeline after generating fragments for the tested object. However, sending graphics hardware all primitives in view frustum to be rendered is inefficient for large scenes such as urban environments where a few close buildings might occlude hundreds of others. Processing all those occluded buildings in the remaining steps of the rendering pipeline is unnecessarily time consuming and useless.

Visibility culling researches are classified as view-frustum culling, back-face culling and occlusion culling. The purpose of view-frustum culling is to discard the primitives outside the view frustum. Back-face culling avoids rendering the geometries facing away from the viewer. However, both of these visibility culling methods do not eliminate obscured primitives inside the view frustum as seen in Figure 1.2.

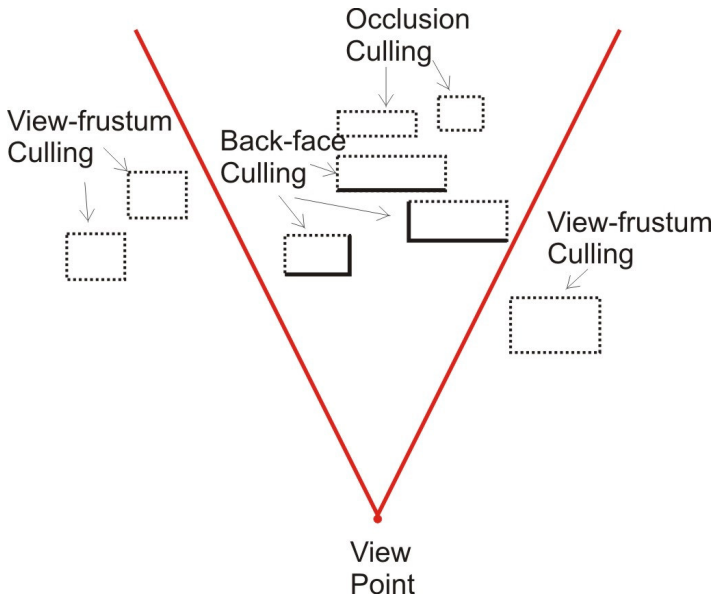


Figure 1.2: Types of visibility culling methods.

The purpose of *occlusion culling* is to avoid rendering primitives that are occluded by the front primitives as seen in Figure 1.2. In this thesis, we focus occlusion culling which aims to identify occluded parts of the scene in the *traversal*-stage of the rendering pipeline. Occluded parts of the scene are culled early in the rendering pipeline and do not go through remaining steps of the pipeline. Thus, we reduce the rendering cost and save up time in an interactive simulation. After this point, we analyze visibility culling methods in terms of occlusion culling.

Visibility culling methods usually utilize scene hierarchy to speed up testing of the objects for visibility. Occlusion culling tests are costly because of dealing with the interrelationship among the objects. The objects in the hierarchy are tested in a top-down manner. In addition, bounding boxes are utilized for complex or closely located or hierarchical objects to reduce the test period. A bounding box is a box which roughly covers object/objects it encloses. Testing

a bounding box is computationally less expensive than testing the surrounded object/objects.

The final target of visibility culling methods is to obtain exact visible set. However, this is too expensive and infeasible for interactive applications. Hence, conservative visibility methods, which overestimate the visible set of primitives, are proposed to compromise between culling accuracy and culling speed [1,5]. This overestimated visible set of primitives is referred to as *Potential Visible Set (PVS)*. PVS includes all visible primitives plus some of invisible primitives. In other words, we may estimate an occluded object as visible, but may never estimate a visible object as occluded in conservative visibility. PVS is computed for either a view point or a view cell/region according to the visibility culling method. Apart from culling accuracy, there have been visibility studies on the location of the viewer (point-based / region-based / cell-based), solution space (2D / 2½D / 3D, discrete / continuous, image precision / object precision) and occluder fusion support [1,6,7].

The importance of visibility culling increases as the virtual scene becomes more complex. The objects and the entire scene get more complex for better realism with the recent developments in technology. Visibility culling aims to decrease the cost of rendering to the complexity of the visible portion of the scene whatever the size of the entire scene is [1,8]. In an ideal manner, a visibility method should be output sensitive which means that its running time should be proportional to the size of the visible set.

The motivation of this study is to develop a walkthrough application in complex distributed urban environment where buildings are the main source of occlusion [2]. The viewer only sees a small part of the city because close buildings usually occlude the ones behind. The methods which are developed for 3D visibility can also be utilized for urban walkthroughs. However, the computation of full 3D occlusion includes a significant overhead since buildings are connected to the ground. If we model the buildings with their

heights as the function of "$z = f(x, y)$", we may say that we have an environment consisting of 2½D buildings. Thus, 2½D calculations should be enough for visibility estimation in urban environments.

Throughout this thesis study, in achieving an efficient visibility, we have developed the following methods which are the major contributions of this thesis.

The first visibility method is a conservative six degrees of freedom incremental occlusion culling method called Delta-Horizon (ΔH) [9]. ΔH method is based on constructing an occlusion horizon (OH), which is a set of connected lines passing just above all visible primitives, for culling the invisible primitives beyond. It computes PVS that is visible from the current viewpoint for each frame on the fly incrementally.

The second visibility method is an adjustable occluder shrinking method which computes PVS with respect to a region. Enhancing ΔH method with the proposed shrinking method enables us to compute PVS once and utilize it in many consecutive frames.

Finally, we propose an efficient visibility estimation method for distributed virtual urban environments called Visibility-based Area of Interest (VbAoI) method. The scene is organized as a graph. Each distributed user is aware of the PVS of static objects for conservative visibility and only interacts with the PVS of dynamically grouped distributed users to reduce the network traffic.

The rest of this thesis is organized as follows. Chapter 2 summarizes the literature survey on visibility in terms of occlusion culling and DVEs. Chapter 3 introduces the proposed object space occlusion horizon culling method and incremental ΔH method. Chapter 4 presents the proposed adjustable occluder shrinking method. Chapter 5 explains the details of the visibility estimation for distributed virtual urban environments. Chapter 6 concludes the thesis and gives the future research directions.

CHAPTER 2

LITERATURE SURVEY

Comprehensive surveys on visibility culling may be found in [1,7,10]. Before reviewing the occlusion culling researches on visibility estimation methods, we initially give the criteria to categorize occlusion culling [1].

2.1 Categorization Criteria in Occlusion Culling

In this section, we summarize the categorization criteria presented in literature for occlusion culling researches [1].

Point vs. Region: This is the major criterion for the visibility methods. Occlusion is computed for either a view point or a region where a viewer moves inside. A visible set computed for a point (i.e., view point) may change when the viewer moves. A visible set is a set of primitives which is seen from a view point or a region. A visible set computed for a region does not change unless the viewer gets out of the corresponding region. It may be utilized in rendering conservatively when the viewer is in the region.

Image Precision vs. Object Precision: Image precision methods perform occlusion culling on the discrete representation of objects after they are transformed to viewing coordinates and fragmented in the rasterization phase. Object precision methods compute occlusion by utilizing the raw objects in 3D real space.

Accuracy of the Output: Culling method is classified with respect to accuracy as exact, conservative, approximate or aggressive. As mentioned before, the target of visibility estimation is to obtain the exact visible set. However, finding out exact visible set is very costly and infeasible for real-time complex virtual environment applications. Therefore, conservative visibility algorithms which overestimate the visible primitives are preferred when the accuracy of visibility is required. Conservative algorithms use the definition of *Potential Visible Set* (PVS) which contains the set of all visible primitives in addition to some of invisible ones. Instead of spending time for testing each primitive's visibility, some of invisible primitives are considered visible and added to PVS. Approximate algorithms overestimate or underestimate the visible primitives while aggressive methods underestimate the visible primitives. Aggressive and approximate visibility methods do not support conservative visibility when the performance is crucial.

All vs. Subset of Occluders: Some methods compute occlusion by utilizing all occluders in the scene while others use only a selected set of occluders such as big occluders in the scene.

Occluder Fusion: Some methods compute occlusion for only individual occluders while others compute occlusion for all overlapped, connected or disconnected occluders.

Dimension: Some methods compute occlusion for a particular dimension such as 2D, 2½D, 3D. Especially the methods proposed for urban walkthrough computes occlusion for 2½D. Occluders in urban environments are buildings connected to the ground. Thus, instead of modeling the buildings in 3D, we may describe them by giving a height value to 2D base geometry of each building.

Convex vs. Generic Occluders: Some methods utilize only convex occluders in contrast to generic occluders for the occlusion estimation.

Special Hardware Requirement: Some methods need special hardware assistance in addition to z-buffer pass.

Need of Precomputation: Some methods need precomputation to select the occluders or compute PVS. Especially region-based methods compute PVS in during a preprocessing phase.

Treatment of dynamic scene: Handling dynamic scenes where there are moving objects in the scene can not be performed in most of the visibility methods since the object hierarchy may change on the fly.

2.2 Point-based Visibility

The proposed incremental occlusion horizon culling method in Chapter 3 is a point based method which is computed from a view point. Point-based visibility methods are computed for each frame since the visible set changes when viewer moves or changes viewing direction. The point-based visibility culling studies which can be roughly classified as geometric, image-based and hardware-based are summarized as follows.

2.2.1 Geometric Point Visibility

Geometric point visibility methods resolve the relations of primitives in object space to compute occlusion.

The method proposed by Coorg and Teller [11,12] makes use of a set of large convex occluders to compute the occlusion in the scene. The method compares two objects, and takes one of them as an occluder and the other as the occludee. The endpoint connecting lines of occluder and occludee are defined as supporting and separating lines which partition the space into regions. In 3D, vertex-edge pairs generate supporting and separating planes instead of lines.

Hudson et al. [13] proposed a method where a set of occluders is chosen dynamically. It computes occluders for each cell in preprocess. When the view point is inside the cell, it computes shadow frustums of the selected occluders to cull the bounding boxes of objects' hierarchy [1].

Bittner et al. [14] improve the Hudson et al.'s method. They combine the shadow frustums of the occluders into an occlusion tree as in [15]. The comparison is reduced to a tree with an O(logN) depth, while taking occluder fusion into consideration.

2.2.2 Image-Based Point Visibility

Image-based methods execute the culling operation on the discrete representation of the image. They fill the image during the rendering of the scene. The subsequent objects are culled away quickly by the already filled parts of the image [1,16].

Greene proposes Hierarchical Z-Buffer (HZB) method, which is based on an octree hierarchy in object space and a z-buffer in image-precision [17]. The z-buffer is defined as a multi-level buffer where the finest layer is the contents of the z-buffer. Each element in all layers holds the furthest z-value. Objects have been tested from coarser to more accurate level.

The Hierarchical Occlusion Map (HOM) method proposed by Zhang [18] is similar to the HZB. HOM keeps opacity information with the z-values of the occluders. The method tests objects for overlap with occluded regions and then compares their depth values. It supports approximate visibility culling. Hence, visible objects through a few pixels can be culled using a user-specified opacity threshold.

Wonka et al. [2,19] proposed a conservative image-based occlusion culling method for urban environments. They use two auxiliary data structures in a regular 2D grid: the scene grid and occluder grid. In each frame, the method

selects a set of occluders from the precomputed occluder grid and renders the occluder shadows to an auxiliary buffer called the cull map. Each pixel in the cull map corresponds to a cell of the scene grid (object space). Visibility of a cell is calculated according to the corresponding pixel value in the cull map. The method supports occluder fusion. The computation time depends on the numbers of pixels and occluders determined in preprocessing. The overhead incurred by copying the frame buffer and visibility traversal increases proportional to the size of the cull map.

Unlike the above methods, former occlusion horizon methods in [20,21]] work in object space and use image plane to find PVS. We discuss these methods in detail at the beginning of Chapter 3 since our proposed incremental method is similar to these methods.

2.2.3 Hardware-Based Point Visibility

Some of the methods use Graphical Processing Unit (GPU) to estimate the occlusion in image space. The drawbacks are sending all the primitives to GPU and reading data from the frame buffer. The latter process is very slow. Hardware vendors have started adapting occlusion culling features into their designs. A feedback loop or flag added to the hardware reports any change in the z-buffer when a fragment passes the depth test [1,22,23]. When computing PVS, the depth test can be optimized to check the bounding boxes or an enclosing k-dop [24], which closely encloses the objects. Hardware vendors also employ occlusion culling methods [25,26] and provide occlusion culling support for common graphics APIs [16,27,28].

2.3 Region-based Visibility

This section includes region-based (from region or view cell) visibility methods.

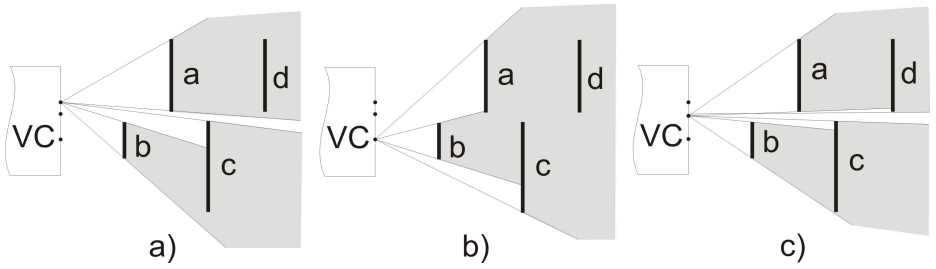


Figure 2.1: An occluded object with respect to two sample points may be visible from a point between them. a-c) Object d is occluded. b) Object d is visible

The basic advantage of region-based visibility methods is that PVS is computed once and may be utilized for a number of frames. However, computing exact or conservative visibility from a region/cell is difficult. Computing visibility from a number of view points in the view cell yields approximated PVS and does not guarantee conservative visibility. When a user moves in the view cell, any primitive which is not in PVS may be visible and causes flickering artifacts in a walkthrough. In other words, view point sampling does not work because a lot of primitives should be visible from a small gap between occluders as seen in Figure 2.1.

One of the important issues in region-based visibility method is supporting occluder fusion where the occluders are connected or separated [1]. Otherwise, PVS will have a lot of primitives when the view cell is larger than the average of occluders. If the sizes of occluders are larger than the view cell, occluders are more effective and PVS is more accurate. For example, smaller objects and their umbrae with respect to view cell are seen in Figure 2.2a. The union of umbras is insignificant without aggregating their umbras (i.e., occluder fusion) as seen in Figure 2.2b,

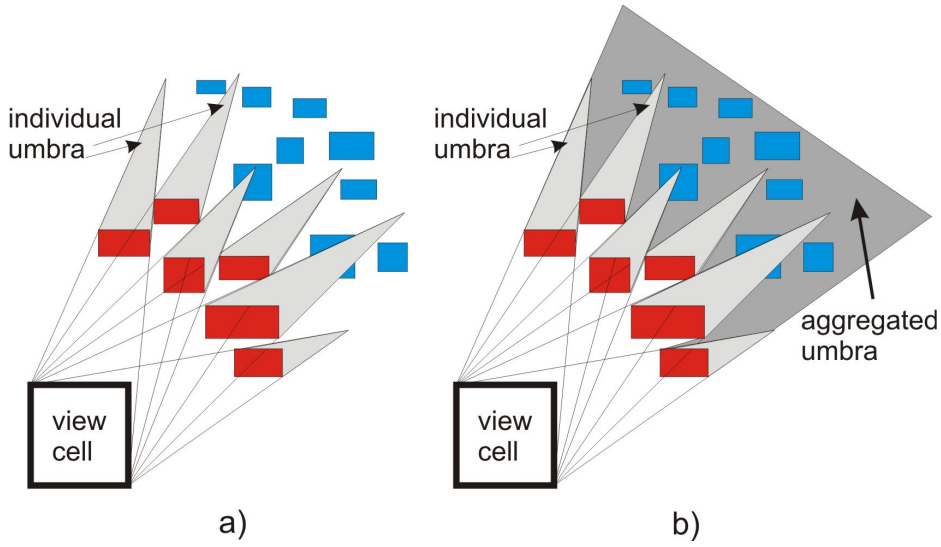


Figure 2.2: a) Umbra of individual occluders b) Aggregated umbra.

Region-based methods are especially important for network applications and disk-to-memory pre-fetching because of its predicting capacity. Prefetching PVS of adjacent cell just before passing into the cell enables a smooth visibility. However, region-based methods have long preprocessing stage, need excessive storage to keep the PVS of regions and are not suitable for dynamic scenes.

Region-based methods proposed for architectural environments are called cell and portal methods. These types of scenes are subdivided into cells. Cells are visible from the openings (such as doors or windows) called portals. PVS is computed for each cell in preprocess and utilized during run-time walkthrough. Airey et al. propose a conservative method for architectural environments [29,30]. This method computes whether a given primitive is visible through a portal. Airey utilizes ray shooting to estimate the visible primitives [29].

Teller's studies on 2D and 3D environments are in Ref. [31]. The scene is divided into convex cells using Binary Space Partitioning (BSP) tree. Opaque

surfaces such as walls are the boundaries of the cells and non opaque portals such as doors are marked on boundaries. They builds adjacency graph for the cells using portals. Cell to cell visibility is determined by using sightlines. If there is a line between two cells, both cells see each others through portals. Adjacency graph is utilized for estimating the cell which a line passes through a portal. Calculated PVS of each cell is used during the run-time walkthrough.

Schaufler et al. [32] proposed a conservative region-based method which works on the discrete representation of space. It determines the occluded regions. This method is effective for a set of occluders and supports occluder fusion.

Durand et al. [33] proposed a method called conservative visibility preprocessing using extended projections. It is an extension of point-based image-precision methods in [17, 18] to volumetric visibility from a view cell. Occluders and occludees are projected onto a plane using the graphics hardware. If the projection of an occludee is in the cumulative projection of all occluders, it is invisible. For conservative visibility, the extended projection of an occluder is underestimating its projection while the extended projection of an occludee is overestimating its projection.

Koltun et al. [34] proposed to utilize virtual occluders for region-based methods. A virtual occluder is a view dependent convex object. It is occluded from all the view points in view cell and utilized as an effective occluder for the corresponding view cell. In other words, virtual occluder is the aggregated occlusion of a set of individual occluders for the corresponding cell. This method also supports occlusion fusion and details of defining virtual occluders may be found in [34]

Wonka et al. [2, 35,36] introduced a region-based method that computes conservative visibility by shrinking occluders. Our proposed shrinking method in Chapter 4 is based on this method and its details are going to be described in the beginning of Chapter 4.

2.4 DVEs and Visibility

In this section, we initially summarize the previous researches on DVEs with respect to real-time contents delivery approach. After that, we mention about the previous researches on 3D streaming which refers to the delivery of the scene contents from network.

2.4.1 DVEs

In literature, the terms of Networked Virtual Environments (NVEs), Collaborative Virtual Environments (CVEs), Massively Multiplayer Online Games (MMOGs) and DVEs are all used for distributed multiplayer real-time simulation systems or games.

Quake III Arena and Diablo II are well known multiplayer online games [37,38]. Their scenes are complex. There are a lot of game objects, increasing the game content size. These type of 3D games in complex scenes require the contents to be obtained [39,40,41]. Because of large contents, game companies prefer to sell the games in a CDROM/DVDROM to end users while some companies prefer to distribute their games from the Internet [42].

A basic research topic in DVEs is to distribute the visible contents to the clients on-time for continuous simulation [39,43]. Loading all the scene content is useless and rendering all the primitives in the scene may not be performed on-time for real-time Distributed Interactive Simulation (DIS) [43]. A client in DVE sees only a small part of the scene because of occlusion. In other words, a distributed client only needs PVS of primitives in his Area of Interest (AoI). The researches on this approach are classified as region-based and interest-based [44]. Some of the methods implemented in several existing DVE systems are given as follows:

Region-based approach was already implemented in the DVE systems of DIVE [45], CALVIN [46], Spline [47] and VIRTUS [48]. Virtual Environments

(VEs) of these systems are divided into regions. A distributed user downloads and loads a region just before entering the region. The size of the downloaded region influences the simulation if it is very large in size. The region definition is usually computed according to spatial hierarchy. Scattering the objects to regions according to their positions is a good way of grouping objects. However, a user needs to load the entire region if he only sees an object in that region. In addition, user may not have conservative visibility if user does not load all the regions in view frustum up to his visible distance.

Google Earth is also a well known application based on region-based approach. A user participating in Google Earth application asks for the 2D satellite images of a region in real-time from the Internet [49].

The interest-based approach focuses on the Area of Interest (AoI) of each user [44,50,51]. In this approach, each user is only interested in the objects in his AoI rather than interested in all the scene contents. A distributed user may navigate in DVE smoothly by only having the scene contents and updates in his AOI. This approach is implemented in NPSNET [50], MASSIVE [52], and NetEffect [53]. AoI approach reduces the amount of transmitted content in runtime. However, there are still problems as follows:

Firstly, AoI computation for a client is not easy. Most of the systems define AoI by the view frustum and viewing distance [44]. This way, systems should accept very long viewing distance and wider view frustum to obtain conservative visibility. Otherwise, user may not obtain conservative visibility because a user may see a lot of primitives from a small gap in the scene. For example, many distant buildings may be visible from a gap between two buildings. Unfortunately, accepting long viewing distance increase the amount of contents to be transmitted.

Secondly, the importance of objects within the AoI is not considered. A user may receive less important or invisible object before the critical and visible ones. Thus, limited bandwidth and time are wasted in runtime.

Thirdly, the geometry information of individual objects is transmitted as a whole to the client traditionally. However, transmitting a large model may easily consume the available network bandwidth for a considerable period of time, affecting the user interactivity.

The interest-based approach may be utilized for grouping the related distributed user to reduce the network traffic by only communicating with the related users in DVEs. There are services which aim to route a communication to the related clients using the AoI approach in DVE applications. For example, High Level Architecture (HLA) establishes common high-level simulation architecture to make possible the interoperability of all types of simulations, models, and C4I systems. HLA is planned to achieve standardization in the Modeling and Simulation (M&S) community and to facilitate the reuse of M&S components [54]. HLA Interface Specification [55] describes a set of services to create federates. The services described in that interface specification is implemented in HLA Runtime Infrastructure (RTI). There are a set of services for Data Distribution Management (DDM) in HLA. These services reduce the network traffic by routing the communications such as attribute updates and interactions to the related federates in AoI.

2.4.2 3D streaming

3D streaming for DVE refers to the continuous and real-time delivery of the 3D scene contents from network. The aim of this approach is to allow distributed users interact with each others without a full download [41,56]. 3D scene contents should be fragmented before transmitting and should be reconstructed and displayed on the client side. The basic difficulty is that PVS of each user is different and should be computed individually. Hu et al. focus on 3D scene streaming for a potentially large DVEs [40,41]. They propose a method for 3D scene streaming on peer-to-peer (P2P) networks identifying the 3D streaming issues as the fragmentation of objects and the prioritization of transmission order. They also propose the method of Flowing Level of Details

(FLoD) which is a scalable P2P framework that supports 3D scene streaming for applications such as The Extensible 3D (X3D) Earth or MMOGs.

The bottleneck of continuous and real-time delivery of 3D scene contents is considered to be the bandwidth rather than rendering or processing power [57]. As a solution, simplification and progressive transmission are proposed [58]. Existing 3D streaming methods are classified as object, scene, visualization, and image-based streaming [40,41].

Hoppe proposed the concept of Progressive Meshes (PM) as an object streaming approach [59]. An arbitrary coarser base mesh which preserves appearance is initially transmitted to the distributed user. User view or interact with this base mesh after its download finishes. Consequently, a number of refinement pieces are sent to the user. User incrementally refines the base mesh and obtains the original mesh in the end. There are many researches on PM such as view dependent PM [60], compressed PM [61], transmission of PM over different networks [62,63,64] and etc.

Scene streaming approach is an extension of object streaming. A set of objects placed arbitrarily in space are streamed with respect to their importance to user. Scene streaming is generally divided into two phases as object determination and object transmission. Schmalstieg et al proposed techniques for scene streaming where each user's visibility is limited to a circular AoI [65]. It is not a good approximation for AoI approach because it does not assures conservative visibility for all view points. If the radius of circular region is small, flickering effects may occur in a walkthrough. If the radius is large, user may need a lot of time to download excessive amount of useless data. There are also researches which focus on objects modeled at different level of details [66]. If switching between different levels of objects is not progressive, data redundancy may occur during the refinement of the object in user side. Cyberwalk [67] utilizes PM to avoid the data redundancy from sending consecutive LoDs. Cyberwalk also adopts caching and prefetching techniques

for optimization. Some of the systems such as in [68,69] utilizes scene streaming for dynamic contents.

Visualization streaming is especially utilized in scientific visualization where large amount of data are processed [40,70,71]. It needs high performance networks and graphics servers to pipeline the processing of large data volumes. It is not applicable for DVEs.

Image-based streaming approach is especially suitable for the clients which have low processing power such as mobile or handheld devices. 3D content of the scene is stored on a server and 2D rendered images generated in real-time are sent to the client to render [72]. Even though taking the 2D image of the DVE in server limits the scalability and interactivity of the system, it is good for using limited bandwidth.

These existing 3D streaming methods focuses on simplification and progressive transmission of contents [40,41]. However, they do not propose how to obtain conservative PVS for each client in DVE. In this thesis study, we focus on estimation of conservative PVS of static and dynamic objects in each client's AoI where AoI is computed with respect to visibility.

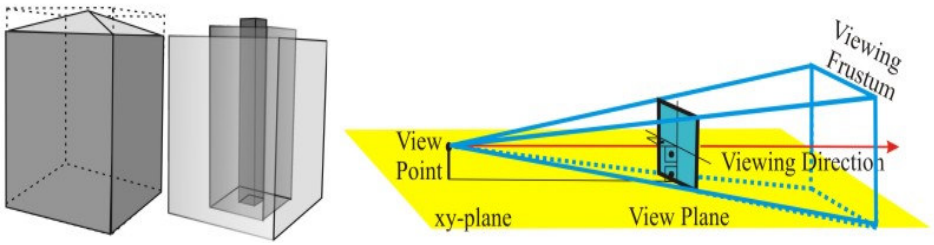
CHAPTER 3

INCREMENTAL OCCLUSION HORIZON CULLING METHOD

Real-time visibility is an important issue in virtual environment applications. Visibility studies focus on culling the invisible primitives to reduce the rendering cost. The basic visibility culling method discards the primitives outside the current view frustum. However, it does not purge obscured primitives inside the view frustum.

The motivation of this study is to develop a walkthrough application in complex urban environment where buildings are the main source of occlusion [2]. The viewer only sees a small part of the city because close buildings usually occlude the ones behind. The methods which are developed for 3D visibility can also be used for urban walkthroughs. However, the computation of full 3D occlusion includes a significant overhead since buildings are connected to the ground. 2½D calculations should be enough for urban environments.

In this chapter, we propose point-based *Delta-Horizon (ΔH)* method for real-time visualization of complex urban environments. It is based on Occlusion Horizon (OH) method proposed by Downs et al.[20].The proposed ΔH method, as in [20], computes the occlusion in 2½D, works on visibility based occluder selection, and supports both conservative visibility and occluder fusion. Besides, it builds OH using polar coordinates in object space which give rise to several advantages. Firstly, ΔH method allows flexible six DoF camera



a) b)

Figure 3.1: a) Left: Building with bounding box. Right: CVPs of a building. b) Viewing direction is parallel to the xy- plane (i.e., ground plane)

movements. Downs' method has only four DoF. Secondly, it accommodates a 360-degree-wide field of view in one pass. This is desired for applications that need wide angle visibility such as radar simulations and panoramic viewing [73]. Thirdly, PVS computed in object space may be utilized without recomputation when the camera zooming is required. Finally, a significant improvement is achieved in updating OH incrementally.

The rest of the chapter is organized as follows. Section 3.1 initially explains the details of the former OH method [20] and then presents the proposed ΔH method in detail. Section 3.2 discusses the performance gain in the empirical results.

3.1 Delta Horizon (ΔH) Method

After discussing the former OH methods in [20,21], we explain the proposed method in detail.

3.1.1 Details of Former OH Methods

Downs et al. use the height fields to determine occlusion in 2½D for urban environments [20]. Each building is modeled with a bounding box and a set of

Convex Vertical Prisms (CVPs) as seen in Figure 3.1a. A bounding box is used for outer hull and CVPs are applied for inner hull to occupy the inside of the building. Additionally, the scene is organized in a 2D quad tree scene hierarchy on the xy-plane. Each quad tree cell keeps the maximum height of any object within the cell or any of its descendants.

A sweeping plane builds OH in a front-to-back traversal. Cells are tested against OH during the sweep. If the entire cell lies below the horizon, it is culled away. If it is not, buildings in the cell are tested. If the building is visible at its minimum distance to view point, it is added to PVS and its CVPs are applied to update OH at its maximum distance. This process ends after all the cells in view frustum are tested.

OH is a series of 2D lines in the view plane which is perpendicular to xy-plane as seen in Figure 3.1b. It is a conservative mask of the space occluded by all buildings encountered in a front-to-back traversal. OH is approximated as a piecewise constant function and represented in a binary tree as seen in Figure 3.2. Each node in the tree has an x range and a y-value for the mask height. The minimum and maximum mask heights of its descendants are stored at every internal node. Testing of a cell or a building is terminated without recursing to the leaves of the tree if its mask is completely below the minimum value or above the maximum value of a portion of the OH.

Lloyd and Egbert [20] adapted OH method to hierarchical terrains. This method also uses a quad tree structure and individual line segments to approximate the edges of the leaf cells instead of CVP to compute OH as in [20].

Overhanging structures and bridges do not contribute to occlusion in the scene, but they will be correctly culled or accepted potentially visible. The drawbacks of these OH methods are summarized as follows:

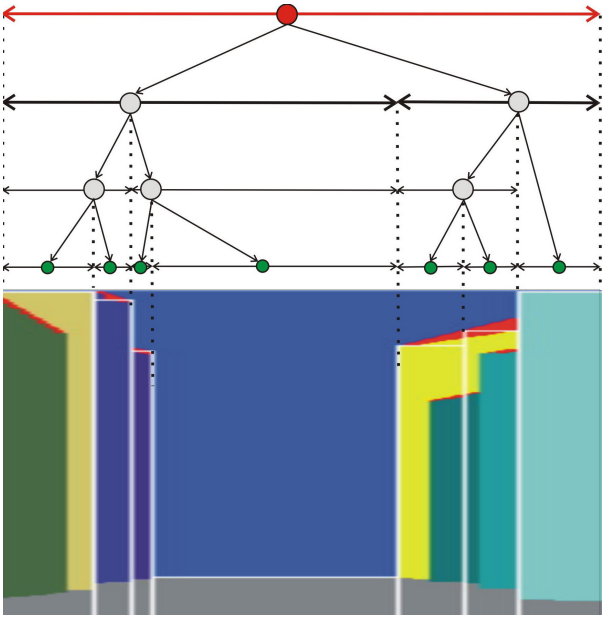


Figure 3.2: Binary tree representation of OH

1) OH can not be built in the view plane when the camera pitches or rolls as seen in Figure 3.3b and c. Thus, these methods have only four DoF.

2) These methods have to build OH and compute PVS for each frame from scratch when the viewer moves.

3.1.2 Object Space OH Method

The motivation of implementing OH in object space is to overcome the drawback of four DoF. This drawback prevents the implementation of OH method in walkthrough application even if it is easy to implement. The main difficulty is how to represent OH in object space. OH is built as a series of 3D lines in polar coordinates[9]. Like the former methods, it is the accumulation of the shadow frustums of CVPs. The basic difference is the computation of

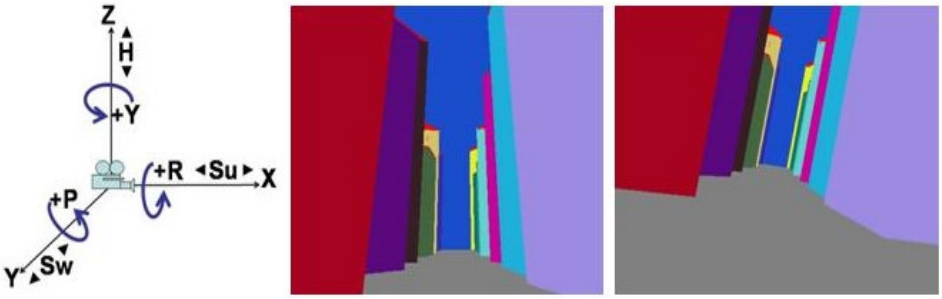


Figure 3.3. a) Six DoF: Surge (Su), sway (Sw), heave (H), roll (R), pitch (P) and yaw (Y). b-c) If camera pitches (P) or rolls (R), OH in the view plane can not be built conservatively in former methods [6,17]

shadow frustum. We summarize how polar coordinates are utilized in the computation of shadow frustums as follows:

The view point is located at the center of polar coordinate system.

In polar coordinates, positive x-axis points to east, positive y-axis points to north, and positive z-axis points up, as seen in Figure 3.4a.

The shadow frustum of a CVP is computed using its visible cross-section and extends to infinity as seen in Figure 3.4a. Shadow frustum is estimated by two horizontal angles and one vertical angle with respect to view point.

Horizontal angles (α_{min}, α_{max}) bound a CVP in both sides. They are measured from east and range between 0 and 360 degrees. To find α_{min} and α_{max} of a CVP, we project CVP to the xy-plane orthogonally as seen in Figure 3.4b. If an occluder shadow spans zero horizontal degree (i.e., extends from 350 degree to 20 degree in polar coordinates), we split it into two parts and test both parts individually.

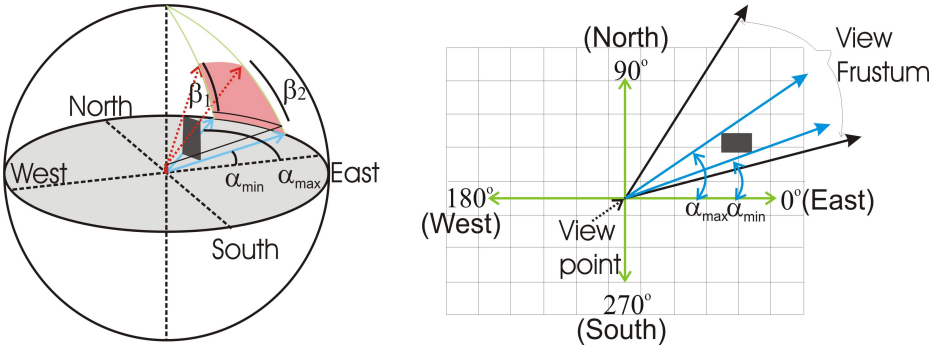


Figure 3.4: a) Occluder shadow in polar coordinates. Horizontal angles (α_{min}, α_{max}) bound a CVP in both sides and measured from the positive x-axis (East). Vertical angle (β-values) bounds the height of a CVP and measured from the viewing direction parallel to the ground plane. b) Top view: OH is built only for the horizontal angle values where view frustum occupies

Vertical angles (β-values) bound the height of a CVP. They are measured from the xy-plane and range between –90 and +90 degrees. The vertices at α_{min} and α_{max} bound the occluder shadow frustum on both sides. The smallest vertical angle of these two vertices is chosen as β_{min} for conservative visibility.

The contributions of the proposed object space method are summarized as follows:

1) It supports six DoF. It does not fail when the camera pitches or rolls since OH is built before viewing transformation.

2) Polar coordinates enable building OH up-to-360-degrees in one pass.

3) Computing PVS before viewing transformation enables zooming without re-computation.

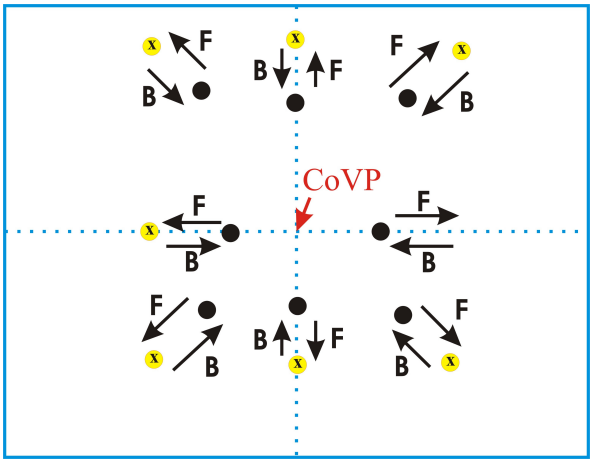


Figure 3.5: Coherence of vertex projection in consecutive frame

3.1.3 Incremental OH Method: ΔH Method

Former methods test all the cells in view frustum and builds OH by the cells in front of OH. However, the invisible cells beyond OH are tested. In the proposed ΔH method, we aim to reduce the cost of testing invisible cells beyond OH. Its motivation is derived from the fact that OH does not change significantly in a consecutive frame during a walkthrough. The proposed ΔH method is implemented in both image and object spaces. From here, we initially explain the coherence of occluders and describe the proposed method in image space.

In image space method, viewing direction is parallel to xy-plane. The primitives in viewing direction project to the Center of View Plane (CoVP) as seen in Figure 3.5. Figure 3.5 shows the coherence of vertex projections in the view plane when the viewer moves forward (F) and backward (B). It is obvious that the consecutive projections move away from CoVP in forward movement and move to CoVP in backward movement.

Table 3.1: Impacts of occluders

User Action	x: part of Current OH	Consecutive OH	
		… in Vertical Axis	… in Horizontal Axis
Forward Movement	∃x above CoVP	moves up	moves away from CoVP
	∃x below CoVP	moves down	moves away from CoVP
Backward Movement	∃x above CoVP	moves down	moves to CoVP
	∃x below CoVP	moves up	moves to CoVP
Lateral Movement	∀x	do not move	move to opposite site
Change in Viewing Direction	∀x	do not shift	shift to opposite site

Figure 3.6 summarizes the coherence of occluders that are taller than the view point's height. They project upper half of the view plane. Visible buildings in forward and backward movements are marked as seen in Figure 3.6d and Figure 3.6f with respect to Figure 3.6e. Figure 3.6g shows the visible buildings (according to left (L) and right (R) movement) in lateral movements. If the viewing direction only changes, the buildings entered the view frustum are colored as blue and green as seen in Figure 3.6h.

We consider how occluders in current OH impact the consecutive OH. The impact of an occluder varies according to its distance in object space and projection position in the view plane. Table 3.1 summarizes the impact of occluders for the user actions below:

Forward movement: The parts of current OH above CoVP move up and its parts below CoVP move down in vertical axis. Simultaneously, they move away from CoVP in horizontal axis as seen in Figure 3.6d. Buildings in front of OH may form OH in consecutive frame regardless of being visible or not, as seen in Figure 3.7a and Figure 3.7b.

Backward movement: The parts of current OH above CoVP move down and its parts below CoVP move up in vertical axis. Simultaneously, they move to CoVP in horizontal axis as seen in Figure 3.6f. An occluded building in front of current OH may be visible as seen in Figure 3.7c and Figure 3.7d. Occluded

buildings behind OH may form OH in consecutive frame as seen in Figure 3.6f.

Lateral movement: All parts of current OH move to opposite side with respect to the movement. The more an occluder is away from the view point in object space, the less its consecutive projection moves in the view plane. Buildings in front of current OH may form OH in consecutive frame.

Changing only viewing direction: If viewing direction changes, new primitives enter the view frustum as seen in Figure 3.6h and OH shifts to opposite side in consecutive frame.

We summarize the abbreviations and the steps of ΔH method depicted in Figure 3.8 as follows:

H*(t)* **:** OH in frame *t*.

H*(t+1)* **:** OH in frame *t+1* (i.e., consecutive OH)

ΔH **:** Delta OH. It shows openings and is utilized to update **H*(t+1)***.

1) Mark the cells in front of H(t) at frame t.

2) Build **H(*t+1*)** by the cells in front of **H*(t)*** from close to far. If an occluded primitive in frame *t* is visible in frame *t+1*, it should be added to the PVS.

3) Calculate "**Δ*H* = H*(t+1)* – H*(t)***". ΔH has some parts risen and some parts lowered (i.e., openings). Subtraction details are as follows: There is no need to check the cells behind the parts of risen ΔH because they have already occluded in frame *t*. Thus the values of these regions in ΔH are extended to maximum values of horizon. Lowered parts of horizon in consecutive frame denote the openings to be checked in object space. We only copy the values of **H*(t+1) to* ΔH** for these regions.

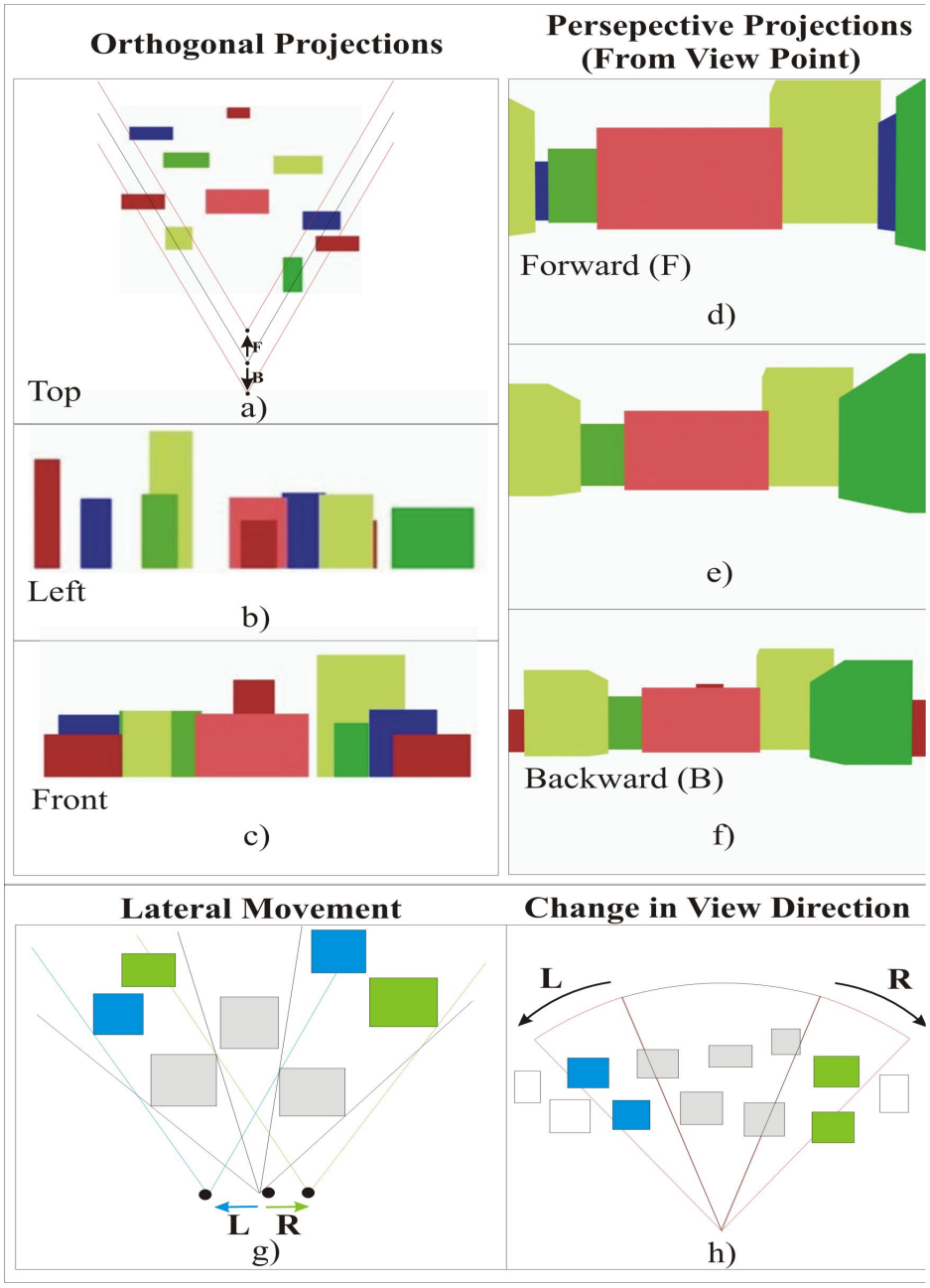


Figure 3.6: Coherence of occluders. Orthogonal projections from a) top, b) left and c) front. d-f) Perspective projections from view point. d) In forward (F) movement. e) Before movement. f) In backward (B) movement. g) In lateral movements. h) In viewing direction changes.

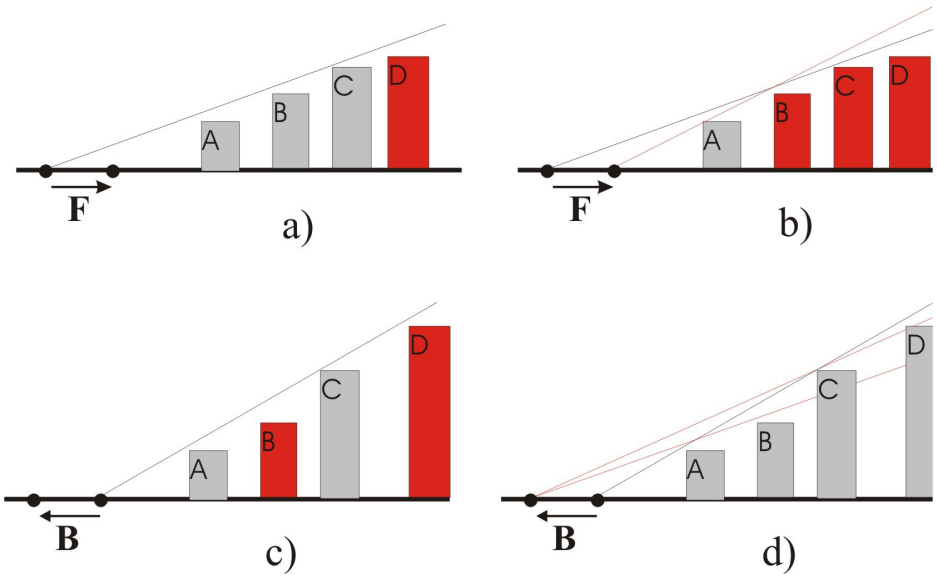


Figure 3.7: a) Before forward (F) movement: A, B, C are visible and C builds OH. D is occluded. b) After forward (F) movement: A is visible and builds OH. B and C become occluded. c) Before backward movement: A and C are visible and C builds OH. B and D are occluded. d) After backward movement: A, B, C and D are visible and D builds OH.

4) Check the cells projecting to the openings of $\Delta\mathbf{H}$ in a front-to-back traversal

5) If the viewer only changes the viewing direction, the cells entered the view frustum should be tested as seen in Figure 3.6h.

After testing all the cells in the openings of $\Delta\mathbf{H}$, $\mathbf{H}_{(t+1)}$ is updated incrementally.

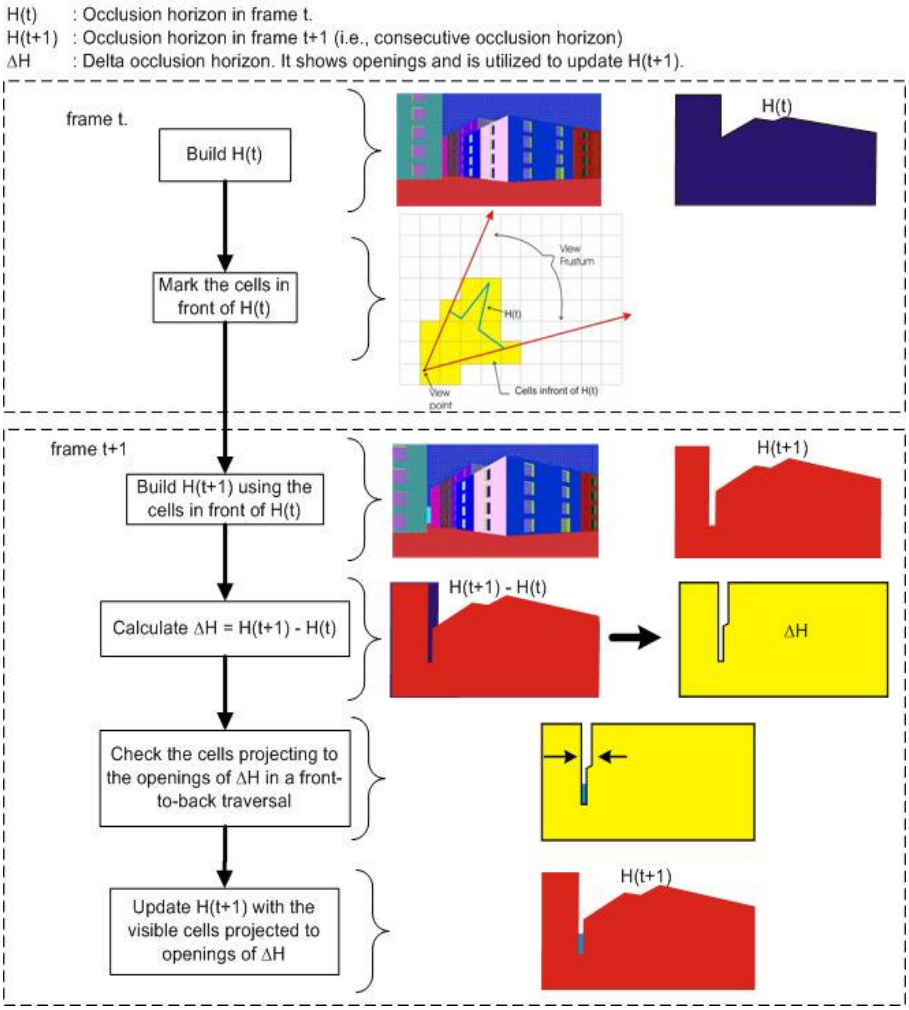


Figure 3.8: Steps of proposed ΔH method

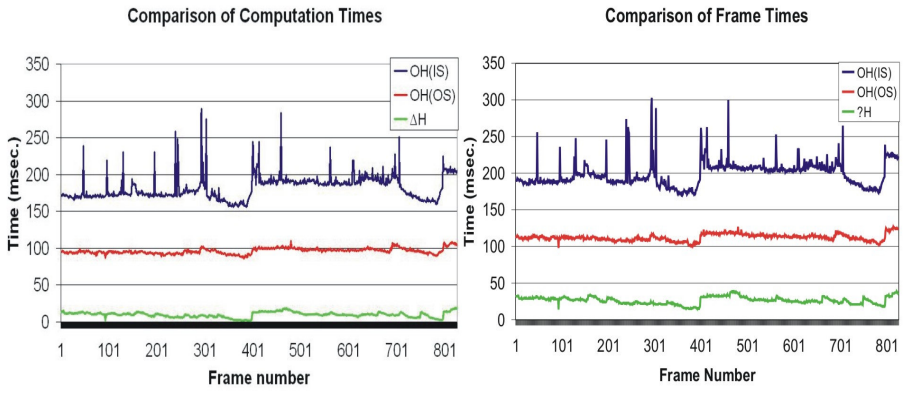


Figure 3.9: Statistics gathered in the testbed a) Comparison of frame times. b) Comparison of OH computation times.

We utilize link list and binary tree data structures together in the implementation. Binary tree data structure enables search the tree in (log n) time. Linked list data structure enables to compute ΔH easily. Thus, we significantly decrease the cost of OH method by reducing the number of cells tested behind OH.

3.2 Results and Discussions

As a testbed, we modeled an urban environment consisting of 160,000 buildings, each of which has 46 triangles. The testbed has 7.36M triangles totally and is organized into 2D quad tree on the xy-plane. The buildings are scattered into the quad tree at the lowest level cell. In the experiments, OH is computed only for view frustum as seen in Figure 3.4b.

We perform our tests on PC Pentium IV 3 GHz. with an nVidia GeForce FX5700LE-256 MB graphics card. We implement the methods in MS Visual C++ 6.0 with OpenGL API. The statistics are gathered on a predetermined walkthrough. In Figure 3.9, we compare the timings of the following culling

Table 3.2: Performance comparison of methods a) Comparison of average times. b) Average speed ups of the proposed methods.

	OH(IS)	OH(OS)	ΔH
Frame Time (msec.)	243.9	113	27.25
Culling Time (msec.)	227.2	96.9	20363

a)

	OH(IS)/OH(OS)	OH(IS)/ΔH	OH(OS)/ΔH
Frame Time	2.16	8.95	4.15
Culling Time	2.34	21.54	9.18

b)

methods: OH culling method in image space (*OH(IS)*), proposed OH method in object space (*OH(OS)*) and proposed ΔH method (Δ*H*) in object space. Figure 3.9a summarizes the frame times which are the total of culling time (i.e., time of building OH and computing PVS) and rendering time (i.e., time of sending PVS to GPU and rendering). We only summarize the culling times to compare the cost of methods as seen in Figure 3.9b.

Table 3.2 compares the averages of times gathered in Figure 3.9 and summarizes the speeds up of the proposed methods. The results show that ΔH achieves 8.95 times speedup in frame time and 21.54 times speedup in culling time over OH(IS). Besides, OH(OS) achieves 2.16 times speedup in frame time and 2.34 times speedup in culling time over OH(IS). The latter speed ups are caused by the difference between estimating the projected coordinates in image plane and angle values in polar coordinates. We utilized gluProject() function to estimate the window coordinates in OH(IS). If there is a cheaper way to estimate the window coordinates in OH(IS), these speedups may reduce. Therefore, we need to compare the times of OH(OS) and ΔH to see the performance gain in ΔH method. It is important to note that ΔH achieves 4.15 times speedup in frame time and 9.18 times speedup in culling time over OH(OS). As a result, ΔH method reduces the costs of frame and culling times significantly.

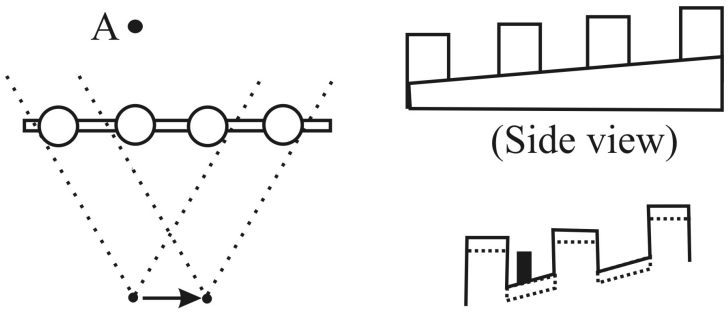


Figure 3.10: a) Top view. Building A becomes unoccluded in lateral motion. b) Side view: Occlusion horizon rises and no cells beyond it are checked, missing the black building.

The performance gain of the proposed ΔH method is caused by reducing the number of testing cells behind OH. In addition, the number of nodes in ΔH tree *(after step 3 in Section 3.3.)* is reduced to 20% at the average. Thus it speeds up testing primitives and updating OH.

There is a trivial constrain for the proposed ΔH method. If user moves more than the width of a building, we can not update the occlusion horizon with a building which has smaller width than the distance moved. This extreme case is depicted in Figure 3.10. The building A is visible in the consecutive frame. However, ΔH method misses it because the horizon is risen as seen in Figure 3.10. This extreme case can be solved by three different approaches: 1) Limiting the movement of a user with respect to the smallest width of the buildings in the environment 2) Updating OH with the buildings which have wider width than the distance of movement. 3) Switching the incremental method to non-incremental method for an instance.

3.3 A Sample Application: Radar Echo Generation

Radars are used for locating objects that are on the line of sight but are not visible because of environmental conditions such as darkness, fog and distance. To solve this visibility problem in real life, radar beams are used to discover the visible cross sections of objects in the environment. Radars estimate visibility from a point in 360 degrees of coverage in reality. Because of this fact, we choose this sample application. In this section, we simulate the echo of mobile radar. Mobility hardens the generation of radar echo in real-time because it is necessary to regenerate the ground echo of the environment in each movement of radar platform. We adapted the proposed object space OH method in Section 3.1.2 for generating real-time radar echo for 360 degrees of coverage in 3D virtual terrain environment [73].

Radar simulation in training increases the experience of personnel and reduces the cost of training. Stationary radars can be simulated by generating radar echo at the beginning and simulating the moving targets on the precomputed image of the environments. Simulation of radars on moving platforms such as aircrafts or ships uses simplified databases (in air to ground radars), tracks only moving objects (in air-to-air radars) or obtains low resolution using limited rays [74,75].

In this section, we focus on echo generation for mobile radar simulation such as vehicle/mast mounted and man-portable ground radars used in battlefield [76]. The coverage area of modular ground radar mounted on moving platform, changes when the platform moves. Real-time simulation of mobile radar requires vast amount of computational resources to regenerate the ground echo (or visible cross sections of objects) in any movement. Radars determine the visible objects in the environment by the propagation of an electromagnetic field. Electromagnetic field exhibits some characteristics of a particle traveling through space and some wave characteristics [77]. Radar simulations are

generally use ray tracing method instead of modeling the characteristics of electromagnetic field. Unfortunately, the cost of keeping the accuracy in ray tracing method increases when the distance of objects in the range increase. To avoid the cost of ray tracing method and increase the performance of the simulation, we utilize object space OH culling method to generate ground radar echo in 3D terrain environment [20,20]. OH culling method uses 2½D features of the virtual environment even if the terrain is modeled in 3D. Thus, the computation complexity of the radar echo generation is reduced from ray tracing to 2½D visibility.

The rest of this section is organized as follows. Section 3.3.1 reviews related work on radar simulations, Section 3.3.2 presents the proposed radar echo generation method in detail. Section 3.3.3 discusses the strengths and limitations of our method.

3.3.1 Related Work on Radars

Radar cross section of an object in the environment is predicted by the propagation of an electromagnetic field [77,78]. Radar simulations model the electromagnetic field which exhibits some characteristics of a particle traveling through space and some wave characteristics [77,79]. Unfortunately, modeling electromagnetic field is very costly for radar echo generation and most of the radar simulations use ray tracing method.

Ray tracing is frequently used to simulate radar echo [77,80]. Fundamentally, rays are propagated trough the scene and the objects intersected with these rays are assumed to be visible. Two approaches are used: forward ray-tracing and backward ray-tracing. For the first approach, rays are sent from the transmitter through the scene. For the second, rays are sent from the objects to transmitter.

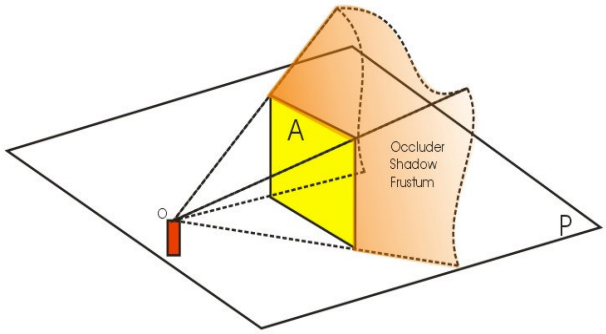


Figure 3.11: Occluder shadow

3.3.2 Radar Echo Generation Method

We generate the ground radar echo of the virtual terrain environment by using object space OH method explained in Section 3.1.2. The objects in the environment are modeled in 3D and assumed to stand on the ground. In other words, objects do not hang in the air. Such an object in the scene can be described by a function "z = f (x, y)", and it is called 2½ dimensional. This assumption enables to use the height of the objects in the construction of occlusion horizon.

Our proposed method computes occlusion horizon as piecewise 3D lines in polar coordinates in object space by accumulating the occluder shadows of visible primitives in a front-to-back order. Polar coordinate system enables this method to be applied to a radar simulation. To understand the proposed method; we describe occluder shadow concept and how it is used in the construction of occlusion horizon, and how to cull the obscured primitives beyond.

Consider a pyramid which is constructed by four rays originated from the viewing point and passing through each corner of the occluding polygon. *Occluder shadow* is defined as the part of the pyramid behind the polygon as

seen in Figure 3.11. We assumed that the polygon is attached to the ground. Thus three edges of the polygon are used to determine its occluder shadows in 2½D. Detailed definition of occluder shadow is given by Wonka [2] In polar coordinates, the occluder shadow of a 2½D primitive is represented by two horizontal angles (α_{min}, α_{max}) on ground plane P and a vertical angle (β_{min}) from the ground plane, as seen in Figure 3.4a.

The proposed method computes PVS of primitives which have a line of sight contact with the radar and culls the obscured primitives. Similarly, the radar echo displays the visible cross sections of all objects in 360 degrees of coverage and looks like the image seen from bird's eye. We inspired from this similarity to simulate the image of radar echo. Thus, we compute the visible primitives in the scene and rasterize them from a camera above to generate the image of radar echo. We modeled the terrain with triangles. To take the advantage of spatial coherence for traversing the triangles efficiently, we scattered the triangles into the cells of regular grid.

Our method computes the PVS based on the assumptions summarized below:

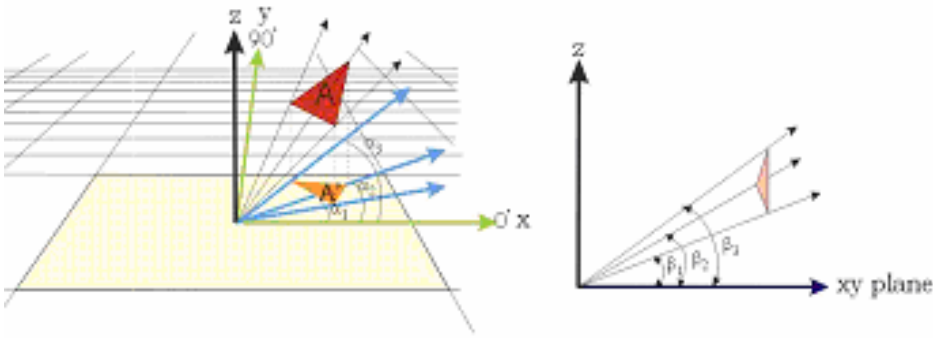
1. The radar is at the center of polar coordinate system.

2. Positive x-axis points to east, positive y-axis points to north, and positive z-axis points up, as depicted in Figure 3.4a.

3. The shadow frustum starts from the visible primitive, extends to infinity and culls the obscured primitives.

4. Horizontal angles (α_{min}, α_{max}) of a visible triangle bound the triangle in both sides and measured from the positive x-axis (East). Thus, they range between 0 and 360 degrees as seen in Figure 3.4a and Figure 3.12a.

5. Vertical angle (β_{min}) of the triangle bounds the upside of the triangle and measured from the ground plane as seen in Figure 3.4a. It ranges between –90 and +90 degrees.



a) b)

Figure 3.12: a) The horizontal component: Orthogonal projection (A`) of triangle A on xy-plane. b) The vertical component: The perspective projection of triangle A on a plane at infinity.

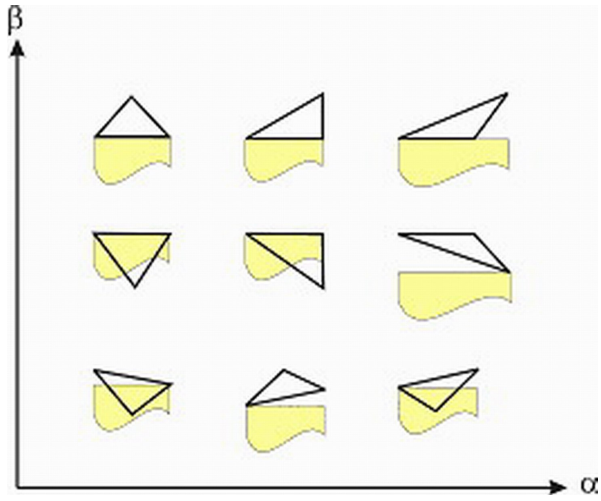


Figure 3.13: Calculating β_{min} by using the projection of the triangle on the sphere at infinity.

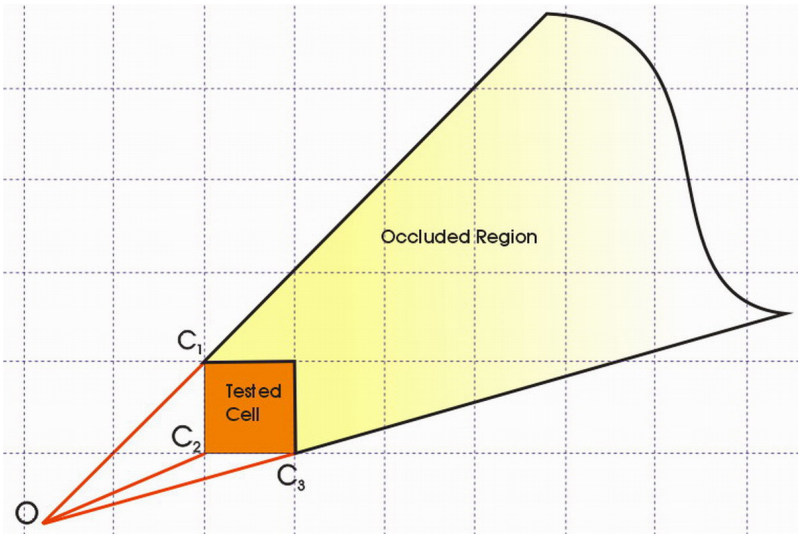


Figure 3.14: Cell test.

To find α_{min} and α_{max} of a triangle, we first locate its orthogonal projection (A`) on the ground plane (xy-plane). As shown in Figure 3.12a, α_{min} is represented by α_1 and α_{max} by α_3.

To determine β_{min} of a triangle, we assume that there is a sphere at infinity as in Figure 3.4a. We project the tested triangle onto this sphere perspectively as in Figure 3.12b. All possible triangle projections on the sphere are depicted in Figure 3.13. The vertices at α_{min} and α_{max} bound the occluder shadow frustum on both sides. For *conservative visibility*, the smallest vertical angle of these two vertices is chosen as β_{min}. For example, consider v_1 and v_3 vertices in Figure 3.12a. β_1 of v_1 is smaller than β_3 of v_3 in Figure 3.12b. β_1 is chosen as β_{min} to support conservative visibility. Thus, occluder shadow of tested triangle is calculated as the region behind the triangle below β_{min} between α_{min} and α_{max}.

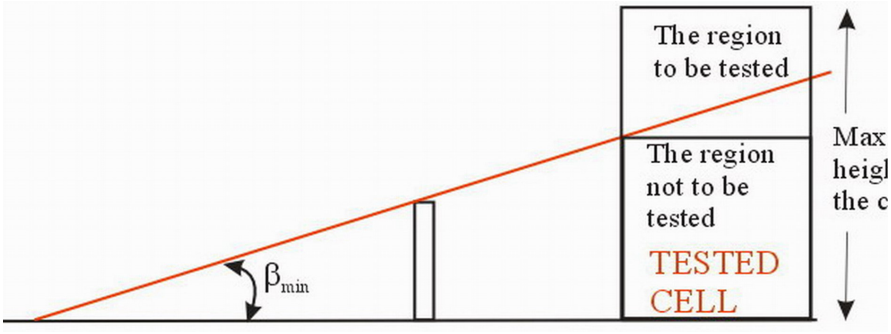


Figure 3.15: The region to be checked in a tested cell.

The occluder shadows of visible triangles are accumulated as occlusion horizon. The method consists of two steps as follows:

In the first step, OH is built by the triangles in the cell of the transmitter. If a triangle faces towards the transmitter, it is added to the PVS and tested against the OH for visibility. If it is above the horizon thus visible, its shadow is used to update the occlusion horizon.

In the second step, we check all cells in the range from front to back. As seen in Figure 3.14, C_1 and C_3 are utilized to estimate α_{min} and α_{max} of the tested cell on the ground plane. C_2 is the closest point of the cell to the viewer and used to estimate β_{min}, assuming that the highest point is at C_2 for conservative visibility. Then, we check whether the tested cell is visible. If the cell is visible, its primitives above β_{min} are added to PVS and used to update occlusion horizon as shown in Figure 3.15.

As a result, we compute the PVS of triangles which are the visible cross sections of primitives in polar coordinate system. We rasterize the computed PVS from a camera above to simulate the image of radar echo in 2D display.

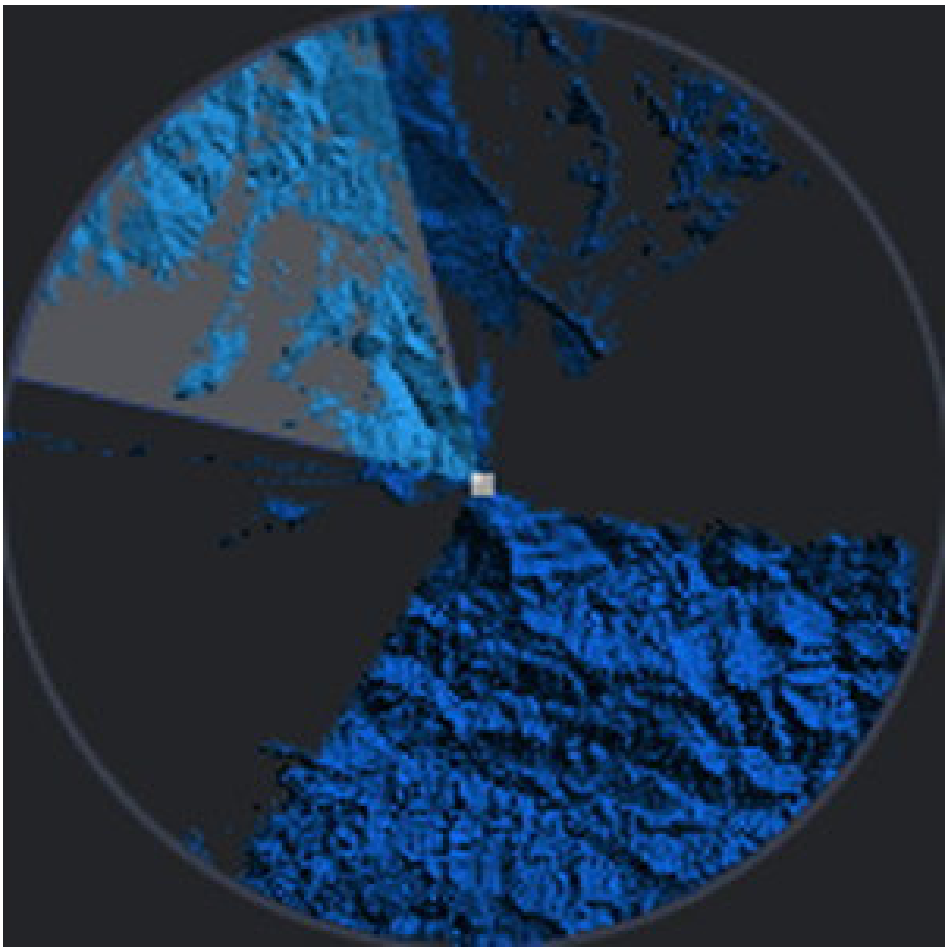


Figure 3.16: Generated radar echo.

3.3.3 Results and Discussion

The terrain model illustrated in Figure 4.16 is about 66 km by 56 km and has ~4.96 million triangles. During the pre-processing stage, terrain is divided into 2D regular grids of various sizes for spatial coherence. The grid sizes of the terrain environment are 30×30, 60×60, 100×100, 200×200, 400×400 cells. If a triangle spans more than one cell, it is referenced by all cells intersecting it. In addition, the highest value of each cell is precomputed and all triangles in each cell are sorted from highest to lowest.

The measurements of the performance in radar echo generation are reported in Table 3.3. These measurements are collected in 1,280×1,024 resolution on a Pentium IV 3GHz PC with a 64 MB NVIDIA GeForce4 MX 400 graphics card. Assuming that the antenna of the radar completes a cycle of rotation in 2 seconds, current performance of the method is sufficient for the grid sizes less than 400×400.

Table 3.3: The measurements of the performance in radar echo generation for 60 degrees of coverage.

Regular Grid Size	Radar Echo Generation Time for 60^o (msec.)
30×30	266
60×60	214
100×100	209
200×200	333
400×400	898

The main strength of the proposed method is its conservativity. Conservative feature of OH method guarantees that all visible primitives are correctly added to the PVS. Thus, it finds all visible primitives in 360 degree of coverage to generate radar echo.

We use regular height field data. Therefore all triangles have the same projection size on the x-y plane. As a result, sizes of the triangles are limited to the accuracy of the terrain data. Depending on this fact, we neglect the error caused by triangles marked as visible because of a very small visible portion.

Besides, cell size is an important parameter for the proposed method. The proposed method culls the triangles cell by cell and computes the PVS more accurately when the cell size is smaller. Unfortunately, reducing the cell size increases the number of cells and proportionally the number of cell tests as seen in Table 3.3. Thus, appropriate grid size of the scene should be determined to obtain desired accuracy of the image at optimum frame rate.

CHAPTER 4

AN ADJUSTABLE OCCLUDER SHRINKING METHOD

Point-based visibility methods estimate visibility for a view point and recompute visibility when the camera moves. These methods can not guarantee constant frame rate for real-time walkthrough applications. Therefore, region-based visibility methods which compute PVS with respect to a region before hand, are proposed [1].

The basic advantage of region-based visibility methods is that PVS is computed once and may be utilized for a number of frames [1]. However, computing exact or conservative visibility from a region/cell is difficult. Computing visibility from a number of view points in the view cell yields approximated PVS and does not guarantee conservative visibility. When a user moves in the view cell, any primitive that is not in PVS may be visible and causes flickering artifacts in a walkthrough. In other words, view point sampling does not work because a lot of primitives may be visible from a small gap between occluders as seen in Figure 2.1.

We enhanced ΔH method with a newly proposed conservative occluder shrinking method. Occluder shrinking method shrinks occluder in preprocess. Shrunk versions of occluders are used in visibility estimation for the samples points on the border of view cell. Shrinking occluders an amount of the half distance between sample points guarantees that an invisible primitive is not visible from a point between sample points as seen in Figure 2.1. This method

is similar to the former shrinking method proposed for region-based visibility [2,35,36]. In former method, a visibility server shrinks all dimensions of occluders ε-distance in preprocess and utilize them to compute PVS for ε-radius circular region around the view point. When the viewer moves inside this circular region, he renders computed PVS and does not update PVS.

Enhancing ΔH method with the proposed shrinking method enables to compute potential visible set once and utilize it in consecutive frames for a circular region during an interactive walkthrough in urban environment. The contributions of the proposed shrinking method are summarized as follows:

- It only shrinks the shadow frustum of a potential visible occluder and computes 30% more accurate PVS.
- Occluder shrinking is performed on the fly and there is no need to store the shrunk versions of occluders preprocessed
- Occluders are selected regarding visibility from the view point on the fly rather than a selection method.
- It is more generic since all operations are performed in CPU without a need of specific graphics hardware.
- Shrinking distance may be adaptively arranged with respect to the speed of user and frame rate on the fly. Hence, it enables to reach a compromise between PVS accuracy and real-time constraints.

The rest of this chapter is organized as follows: Section 4.1 initially explains the details of the former shrinking method [2,35,36] and then presents the proposed adjustable shrinking method in detail. Section 4.2 summarizes and discusses the empirical results.

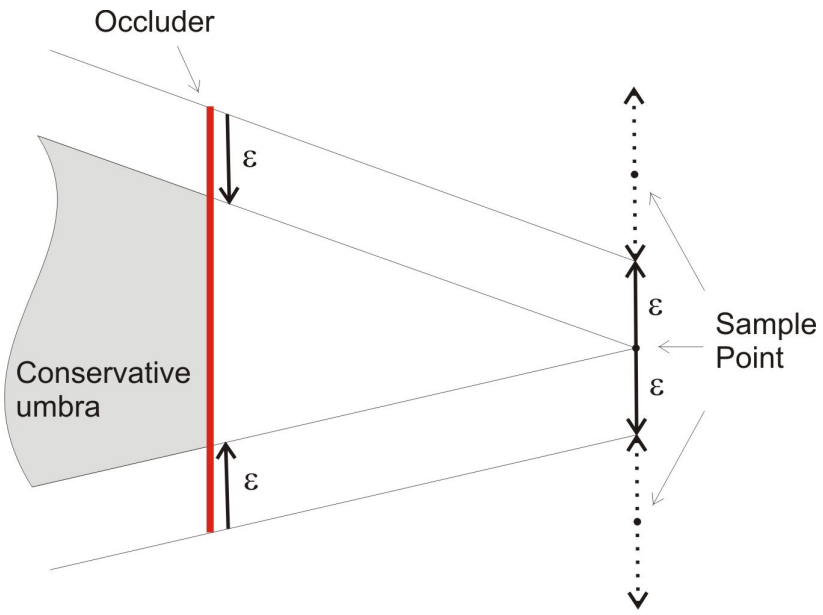


Figure 4.1: Occluder Shrinking.

4.1 Adjustable Occluder Shrinking Method

We explain the proposed new occluder shrinking method for 2½D environments in detail after we discuss the former occluder shrinking method [2,35,36].

4.1.1 Details of Former Shrinking Method

Wonka et al. proposed a method based on the intersection of the occluder shadow umbras calculated for a set of discrete point samples placed on the view cell's boundary [2,35]. The umbra of a sample point is calculated by the accumulation of occluder shadows. The main difficulty is that an occluded object with respect to two sample points may be visible from a point between them as seen in Figure 2.1. Wonka proposed to shrink occluders ε-distance,

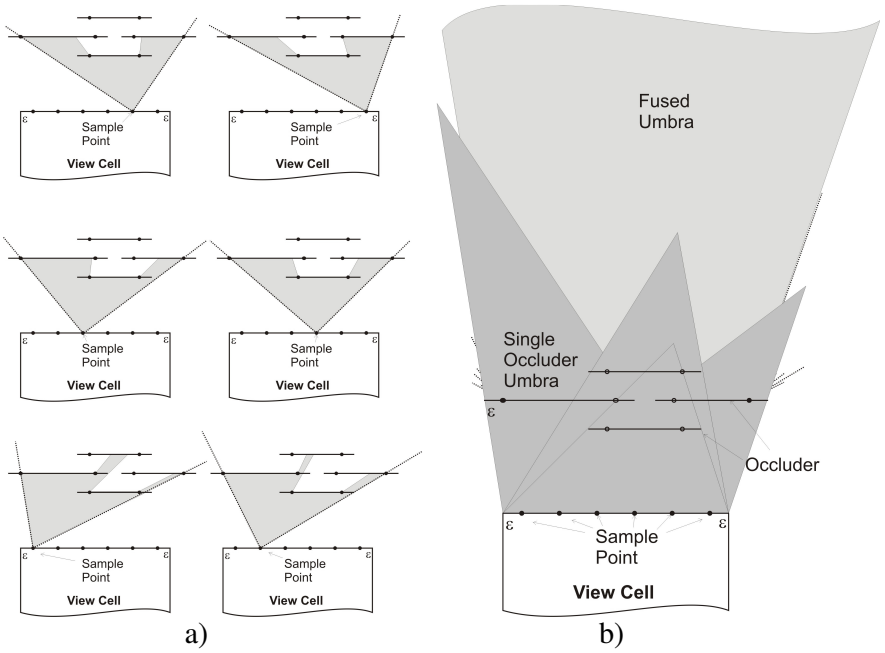


Figure 4.2 a) Sampling of the occlusion from six sampling points. b) The fused umbra from the six points is the intersection of the individual umbrae.

which is the half distance of two consecutive sample points on the view cell's boundary as seen in Figure 4.1. A smaller umbra is built by shrinking an occluder ε-distance. It guarantees that an occluded object does not visible from e-radius around the sample point as seen in Figure 4.1. An object is occluded for the whole view cell if it is contained in the fused umbra of the shrunk occluders for all sample points on the view cell's boundary as seen in Figure 4.2. The proposed method also supports occlusion fusion as seen in the Figure 4.2.

Wonka adapted shrinking method to a point-based visibility method in Ref. [36]. This method enhances a point-based visibility method with the shrinking occluder approach which is proposed for region-based visibility. A visibility server computes PVS for ε-radius circular region around the current view point and sends it to the client. Client renders the computed PVS for consecutive

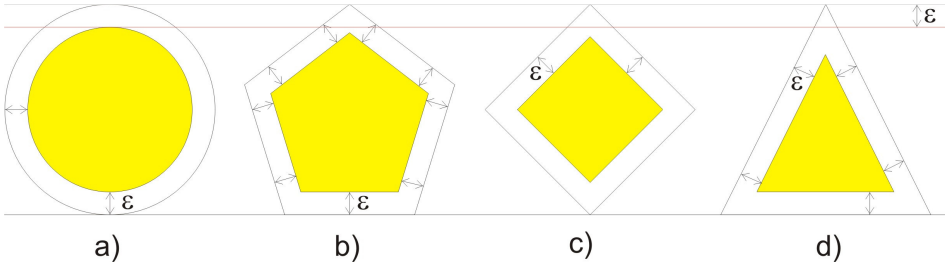


Figure 4.3: Shrinking occluders ε-distance in preprocess from top view. Occluders are shrunk in 2D. Shrunk occluders are colored yellow. Occluder shrinkings are compared with two horizontal lines at the top. The distance of two lines is equal to ε-distance. a) Optimum shrinking occurs in a circle. b-d) Amount of excessive shrinking increases in parallel to the sharpness of the edges of primitives as seen from b to d.

frames when he moves in ε-radius circular region. Thus, this method allows asynchronous processing of conservative visibility and display operations.

In this method, 2½D occluders are shrunk in 2D and PVS is computed with respect to the highest point where the camera can move up [35]. A selected set of occluders are shrunk in preprocess. A visibility server utilizes the stored shrunk occluders to compute the combined shadow frustum and PVS. The minimum distance between a shrunk occluder and original occluder should not be less than ε-distance for conservative visibility as seen in Figure 4.3.

This shrinking method uses graphics hardware to compute PVS [2,35,36]. The disadvantage of using graphics hardware is that it requires read access to the frame buffer for each occluder test. To reduce the total cost of reading frame buffer, they select a set of occluders that is most likely to occlude a large part of the scene. However, missing small occluders may create holes that make additional objects visible and cause an increase in the number of objects in PVS.

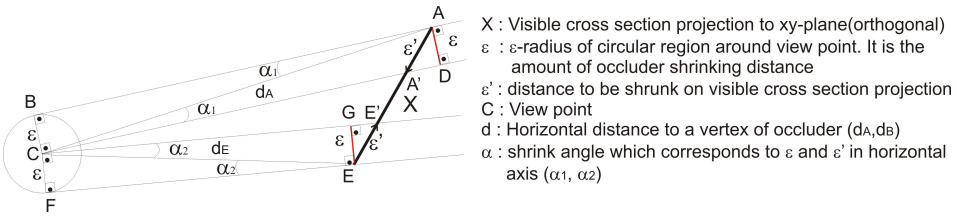


Figure 4.4: Shows the amount of shrinking on the visible cross-section (shadow frustum) of an occluder. We calculate shrink angles (α_1 and α_2) which correspond to shrink distances (ε and ε's) for two vertices (A and E) of an occluder visible cross-section

4.1.2 Adjustable Shrinking Method

The proposed method is based on shrinking the shadow frustum (i.e., visible cross section) of an occluder on the fly. In former methods, occluders are shrunk from all dimensions in a preprocessing step. The distance to be shrunk for each vertex of an occluder should be equal or more than ε-distance as seen in Figure 4.3. It is seen that shrinking gets more if the vertex which intersects two edges gets sharper. Otherwise it does not support conservative visibility in former method. The proposed method shrinks occluder shadows instead of occluders on the fly. We shrink an occluder shadow ε-distance exactly although its shadow frustum changes with respect to the view point. Therefore, the proposed method computes more accurate PVS supporting the conservative visibility.

We implemented the proposed shrinking method with ΔH method. The proposed method adaptively shrinks the visible cross section projection (i.e., shadow frustum) of an occluder in angular values. The shrinking angles are computed for horizontal and vertical axes. Horizontal shrinking angles always correspond to ε-distance as seen in Figure 4.4 while the vertical shrinking

angle is a variable shrinking distance which guarantees conservative visibility in vertical axis. We previously calculated horizontal angles (α_{min}, α_{max}) to update OH as explained in Section 3. Therefore, we initially need to calculate shrinking angles for α_{min}, and α_{max} as seen in Figure 4.4.

[BC] is parallel to [AD] and [CF] is parallel to [GE] as seen in Figure 4.4. Their lengths are equal to ε-radius as seen in Eq.1 and Eq.2. In addition [BC] is perpendicular to [AB] and [CD]. [CF] is perpendicular to [FE] and [CG] as defined in Eq.3. It is seen that ABCD and EFCG are rectangles. Angle ACD ($\angle ACD$) is equal to angle BAC ($\angle BAC$) and denoted by α_1. $\angle ECG$ is equal to $\angle CEF$ and denoted by α_2. Hence, the calculation of horizontal shrinking angles (α_1 or α_2) is given in Eq.4 and Eq.5. Consequently, we calculate the shrunk horizontal angles (α_{mins}, α_{maxs}) as seen in Eq.6.

$$[BC]//[AD], |BC| = |AD| = \varepsilon \quad \text{Eq 1}$$

$$[CF]//[GE], |CF| = |GE| = \varepsilon \quad \text{Eq 2}$$

$$[BC] \perp [AB], [BC] \perp [CD], [CF] \perp [FE], [CF] \perp [CG] \quad \text{Eq 3}$$

$$\angle ACD = \angle BAC = \alpha_1 = \arcsin\left(\frac{\varepsilon}{|AC|}\right) \quad \text{Eq 4}$$

$$\angle ECG = \angle CEF = \alpha_2 = \arcsin\left(\frac{\varepsilon}{|CE|}\right) \quad \text{Eq 5}$$

$$\begin{aligned} \alpha_{\max s} &= \alpha_{\max} - \alpha_1 \\ \alpha_{\min s} &= \alpha_{\min} + \alpha_2 \end{aligned} \quad \text{Eq 6}$$

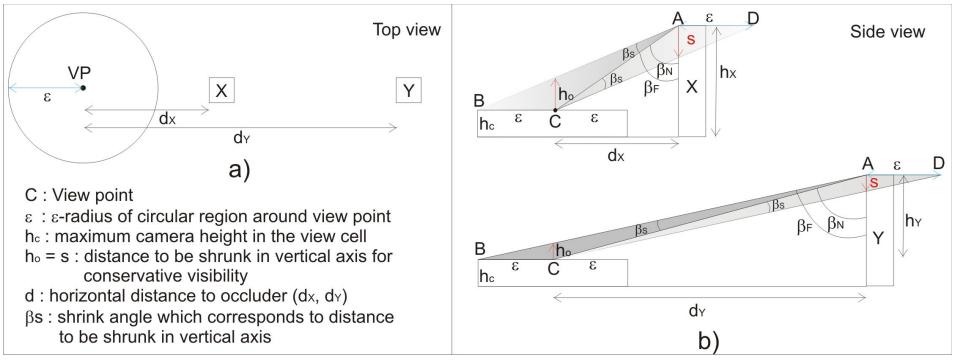


Figure 4.5: a) Top view: PVS are computed for ε-radius circular region around the current view point. b) Side view: around the view point. It is necessary to shrink an occluder as the distance of s which equals to h_o.

To support conservative visibility for 2½D scenes, it is necessary to shrink an occluder shadow in vertical axis and calculate the shrunk vertical angle (β_{mins}). Figure 4.5 shows the relation between circular region around view point and occluders (building X and Y) from top and side views. A shadow frustum should be shrunk as the shrinking distance (*s*) in vertical axis as seen in Figure 4.5b. Shrinking distance (*s*) changes with respect to the distance of occluder. Thus, the proposed method shrinks an occluder shadow adaptively as seen in Figure 4.5b. *[BC]* is parallel to *[AD]* and their lengths are equal to ε-radius as given in Eq.7. AB*CD* forms an equilateral quadrangle as seen in Figure 4.5b. The amount of *s* is equal to vertical distance (h_o) of view point (*C*) to *[AB]* as given in Eq.8. As seen in Figure 4.5b, h_o changes with respect to corresponding occluder height (h_X/h_Y) and distance (d_X/d_Y) from the view point. We calculate vertical shrinking angle (β_S) which corresponds to *s* and h_o. Thus, we calculate β_{mins} which is the shrunk vertical angle, as summarized below:

$$[BC] // [AD], |BC| = |AD| = \varepsilon \quad \text{Eq 7}$$

$$(h_o \perp [BC], s \perp [AD], \angle ABC = \angle CDA) \rightarrow h_o = s \quad \text{Eq 8}$$

$$\beta_F = \arctan\left(\frac{\varepsilon + d_X}{h_X - h_C}\right) \quad \text{Eq 9}$$

$$\beta_N = \arctan\left(\frac{d_X}{h_X - h_C}\right) \quad \text{Eq 10}$$

$$\beta_S = \beta_F - \beta_N \quad \text{Eq 11}$$

$$\beta_{\min s} = \beta_{\min} - \beta_S \quad \text{Eq 12}$$

If the tested occluder's height is less than the camera's height as seen in Figure 4.6, shrunk vertical angle is calculated by changing the Eq.9 and Eq.10 with Eq. 13 and Eq.14 defined as follows:

$$\beta_F = \arctan\left(\frac{d_X}{h_C + h_X}\right) \quad \text{Eq 13}$$

$$\beta_N = \arctan\left(\frac{d_X - \varepsilon}{h_C + h_X}\right) \quad \text{Eq 14}$$

We calculated β_{min} as the smallest vertical angle in OH method as explained in Section 3. It ranges from -90^o to $+90^o$. We may use Eq.15 instead of Eq.10 for the occluders higher than camera. We may also use Eq.16 instead of Eq.13 for the occluders lower than the camera's height.

$$\beta_N = 90^o - \beta_{\min} \quad \text{Eq 15}$$

$$\beta_F = 90^o + \beta_{\min} \quad \text{Eq 16}$$

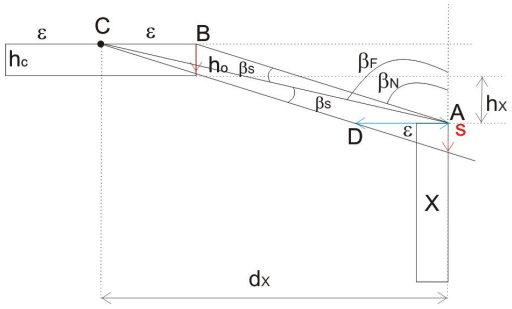


Figure 4.6: Calculation of shrunk vertical angle of a building which has less altitude than the view point (C).

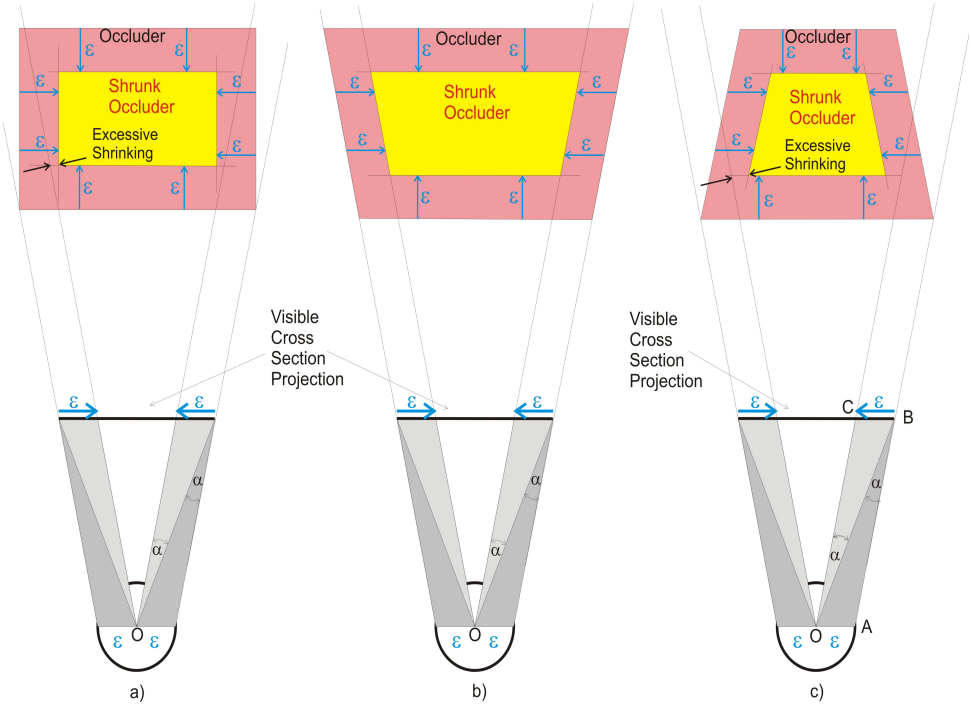


Figure 4.7: Comparisons of occluder shrinking and its visible cross-section shrinking. Occluders are red and its shrunk versions are yellow. Blue colored arrows show the shrinking on the visible cross section projection. a-c) Shrinking occluders ε-distance in 2D causes excessive shrinking. b) Optimum occluder shrinking. c) Calculating ε-distance on the visible cross section projection. (i.e., Angle (BOC) = Angle (ABO)).

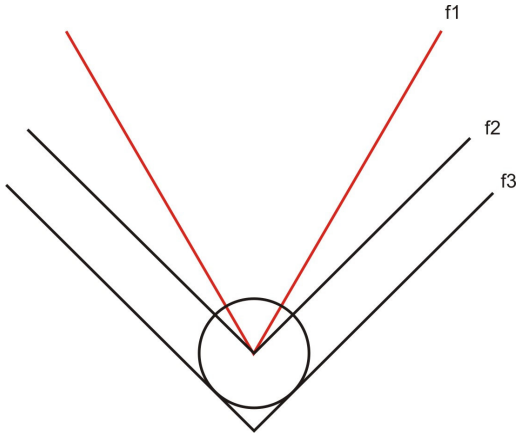


Figure 4.8: Extending view frustum. f1 is the original frustum. f2: is the extended frustum in angular values to utilize the computed PVS in extended angle values. f3: extended frustum to show the e-radius circular region which PVS is computed for.

As a result, we utilize α_{mins}, α_{maxs} and β_{mins} for each occluder to update OH with the proposed adjustable shrinking method.

PVS is only computed for a view point and its directional viewing frustum. Viewer does not need to recompute PVS when he moves in ε-radius circle around view point without changing viewing direction. ε-radius may be adjusted by the speed of user. If ε-radius is very small, we compute more accurate PVS but it is utilized for less number of frames. If ε-radius gets larger, PVS gets coarser and is utilized for more consecutive frames. When client changes the viewing direction, there are three ways to use the computed PVS for consecutive frames: 1) Compute OH and PVS for the angle values which enter the view frustum incrementally as explained in Chapter 3. 2) Instead of computing OH and PVS just for the view frustum, we enlarge the view frustum with respect to the maximum speed of turning around [36] as seen in Figure 4.8. For example, assume that view frustum is 60 degrees and user may turn

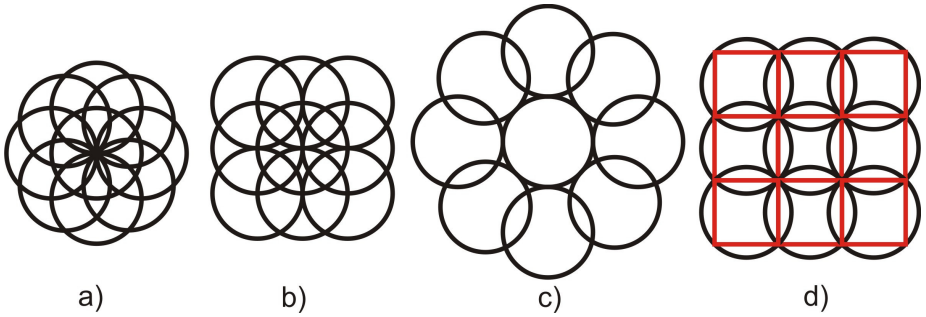


Figure 4.9: Shows the possible view cell interactions. a-b) Excessive overlaps c) There are holes between circles d) Optimum overlap.

around at the speed of 0.5 degree/frame. Enlarging the view frustum to 90 degrees of coverage guarantees that PVS does not need to be recomputed at least 30 frames in changing viewing direction. 3) We may utilize both ways together as follows: Firstly, enlarge the view frustum with respect to the speed of turning around. Secondly, compute OH and PVS when user needs to turn around more than the enlarged view frustum.

PVS is computed for ε-radius circular region around a view point. When viewer moves in the corresponding region we have enough time to compute the PVS for the neighbor regions. The problem is where the center of each neighbor circular regions is. In other words, which view position we should have to compute PVS for .

We summarize the possible view cell (i.e. ε-radius circular region around view point) interactions in Figure 4.9. If the distance between the centers equals to ε-radius, there are too much overlap as seen in Figure 4.9a-b. It costs too much repetitive computations. If the distance between centers equals to two times of ε–distance, holes occur between circles as seen in Figure 4.9c. We propose a new dynamic spatial partitioning approach for the proposed shrinking method as seen in Figure 4.9d. It is seen that the amount of overlap is optimum and there is no hole between circles. This approach enables to estimate the potential

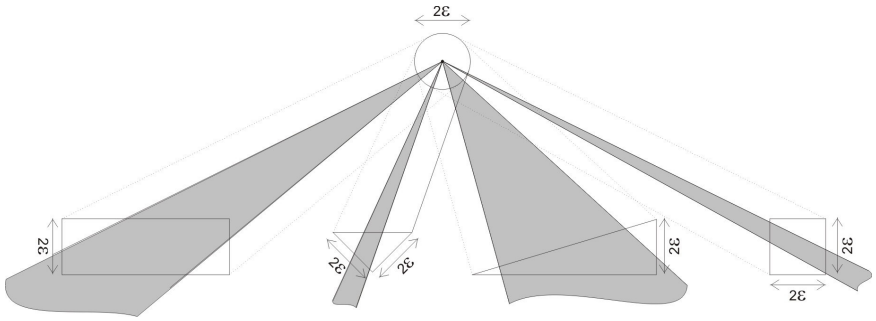


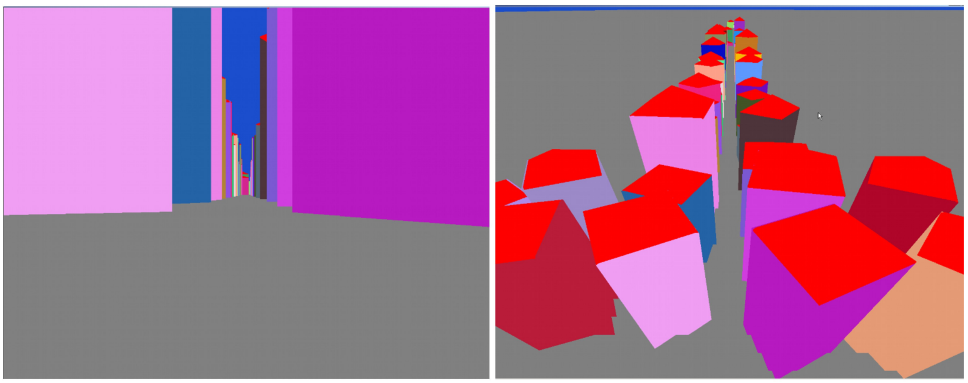
Figure 4.10: Occluders which have smaller width than two times of ε-distance can not be shrunk in former method and do not update fused umbra. The proposed approach shrinks the occluder shadow of these objects and utilizes them to update the fused umbra (i.e.,OH).

next view point positions which we should compute PVS for. Thus, PVSs are predicted for neighbor regions according to the viewer movement direction when the viewer is still in the previous circular region.

4.2 Results and Discussion

The proposed method computes more accurate PVS by only shrinking shadow frustum of an occluder. Occluders are selected regarding visibility from the view point on the fly rather than a selection method in preprocess. It only shrinks the shadow frustum (i.e., visible cross section projection) of each potential visible occluder on the fly although the shadow frustum of an occluder changes with respect to the view point.

The proposed method computes more accurate PVS as explained follows: 1) There is no excessive shrinking of an occluder as seen in Figure 4.3 and Figure 4.7. We compare the proposed shrinking method with the former shrinking method in Figure 4.7. It is seen that the amount of shrinking exactly equals to



a) b)

Figure 4.11: a) Viewer navigates in the street and sees only the potentially visible buildings. b) The buildings in PVS are seen from the camera above the viewer (In wide angle). There should be buildings in all the gray area without culling.

ε-distance in the proposed method. However, shrinking distance is equal or more than ε-distance in the former method as seen in Figure 4.3 and Figure 4.7. 2) The objects which have smaller width than two times of ε-distance can not be shrunk in former method. They should be added to PVS without updating the fused umbra of occluders (i.e., OH). It causes an increase in PVS. Some of them may have wider visible cross section projections than two times of ε-distance as seen in Figure 4.10. The proposed method shrinks their visible cross sections and utilizes them to update OH as colored gray in Figure 4.10.

The proposed method is implemented in a multithread environment. We utilized the testbed described in previous chapter. Visibility thread only loads a header file which has bounding boxes and CVPs of all buildings in a quad tree hierarchy. It does not load the data of buildings into memory. It only computes the PVS of buildings using this header file. Another thread loads the buildings in PVS to shared memory in an on-demand loading manner. Render thread just

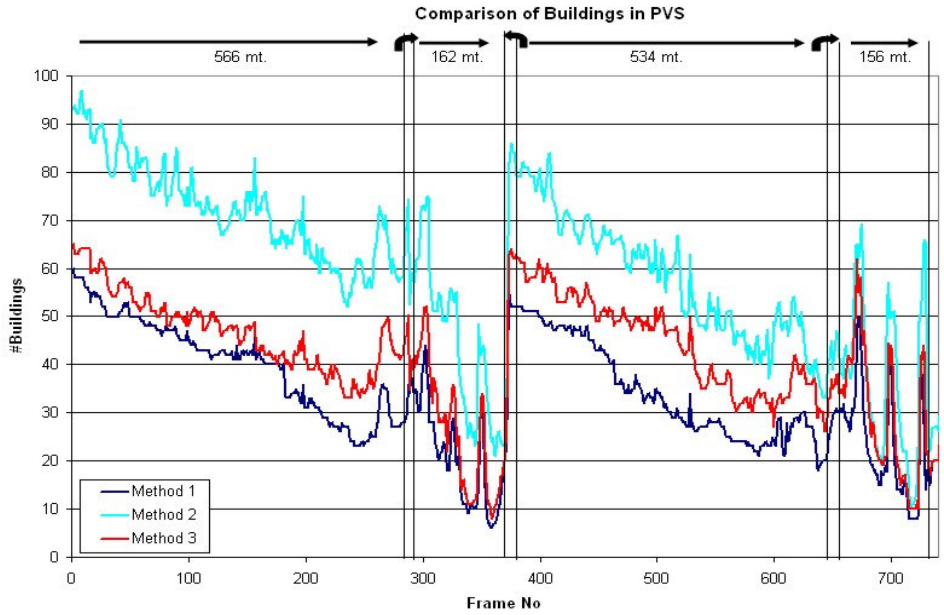


Figure 4.12: Timing gathered during a path in a walkthrough in visibility thread

renders the buildings in the corresponding PVS. Instead of visibility thread, we may use a server as a visibility server and only send the computed PVS to the viewer [36].

Our aim is to develop a real-time walkthrough application which supports 30 frames in a second. We assume that a pedestrian walks 5 km in an hour. It means that viewer moves 1.4 mt. in a second. The 2 meters step allows computing PVSs for $\sqrt{2}$ -radius circle around the view points. Shrinking $\sqrt{2}$ -distance is enough to compute PVS once and utilize it for 30 frames in a second. Rendering of a frame is related to the number of buildings in PVS. Our empirical results show that the rendering time for a frame is not more than 15 msec. during the path.

Viewer navigates through the streets and only sees the buildings along the street conservatively as seen in Figure 4.11a. Figure 4.11b shows the buildings

in the PVS which are seen from the camera above the viewer in wide angle. In normal circumstances, there are buildings on all the gray area in Figure 4.11b.

Performance statistics are gathered on the predetermined path for the following three methods: *Method 1*: OH method with no shrinks. It is utilized to estimate the optimum PVS for 60 degrees of viewing frustum. The computed PVS is only valid for view point. *Method 2*: OH method with former shrinking method for 90 degrees of vieiwing frustum[36]. *Method 3*: OH method with the proposed shrinking method for 90 degrees of vieiwing frustum. The empirical results gathered during a path in a walkthrough are represented in Figure 4.12 and the average times are summarized in Table 4.1.

The details of the path are as follows: Visibility thread computes the PVS for 90 degrees of viewing angle for Method 2 and Method 3. We aim to use the computed PVS for ε-distance in movements and 15 degrees in changing viewing directions to both sides. To compare the number of buildings increased in Method 2 and Method 3, we also computed the PVS of Method 1 for 60 degrees of viewing angle as a reference. Visibility thread computes PVS in a step of 2 meters (mt.) in forward movement and recomputes PVS when viewer changes viewing direction more than 7 degrees. At the top of Figure 4.12, forward movements are represented by arrows which show the distances traveled below. Changing viewing directions are represented by curved arrows which signs the turning direction. The number of buildings in the PVSs of all methods is computed for a step (i.e., a frame). A PVS computed for a step is utilized for 30 frames by the viewer.

The scenario details are as follows: In the first forward movement (566 mt. long), the number of buildings in PVS reduces when he reaches the ends of the street. On the end of the street, there is a square which is 170 mt. long diagonally. Viewer enters this square from one of its corner, changes his direction to diagonal and moves forward 162 mt. more. He comes to the end of square. The buildings seen from this point are very close and occlude others. In

Table 4.1: Comparisons of Methods.

Method	Viewing Angle	Buildings	Triangles	Increase in PVS
Method 1	60º	33,7	1550	0%
Method 2	60º	56,8	2612	68%
Method 3	60º	39,6	1822	17%
Method 1	90º	35,7	1640	6%
Method 2	90º	59,1	2717	75%
Method 3	90º	41,7	1916	24%

other words, the number of visible buildings is less than the other frames. Viewer turns left hand side 56 degrees to enter a long street (550 mt.). It is noticed that the viewer may see long distances in the street after the first half of its turn. The number of buildings increases for the corresponding frames. Similar path is followed once more for the street which the viewer is directed.

The statistics in Table 4.1 are gathered for 60 and 90 degrees of viewing angles respectively. The statistics for 60 degrees are computed as a reference. In former method (Method 2), PVSs have 75% more buildings for 90 degrees of viewing angle. In the proposed method (Method 3), PVSs have only 24% more buildings. Widening the view frustum for 15 degrees in both direction costs about 7-8% more buildings in all methods for this path. Its reason is that front buildings occlude behind and view frustum culling discards the buildings in perpendicular streets to moving direction. As a result, the proposed new method computes 30% less buildings in PVS than the former method [36].

Additionally, the proposed method is more generic since all operations are performed in CPU without a need of specific graphics hardware. All the computations are summarized in Eq.1 to Eq.16. It does not bring a significant overhead because these calculations are only performed for potentially visible buildings. Occluders are selected regarding visibility from the view point on

the fly rather than a selection method in preprocess. Visibility-based occluder selection is the main purpose of visibility methods. Thus, the other occluder selection methods (such as estimating large occluders in the scene) are the subset of visibility-based occluder selection method. The proposed method has no preprocess for shrinking occluders and does not need to store the shrunk versions of occluders. Finally, ε-distance, which is adjustable with respect to the speed of viewer and frame rate on the fly, may be a good way to obtain real-time conservative visibility for complex urban environments.

CHAPTER 5

EFFICIENT VISIBILITY FOR DISTRIBUTED VIRTUAL URBAN ENVIRONMENTS

The number of real-time simulations in complex Distributed Virtual Environments (DVEs) increases in parallel to the recent developments in technology. Multiple users, located in geographically different places, interact with each other in real-time [43]. DVE provides users with a sense of realism by incorporating 3D graphics and sound to create an immerse experience. For a smooth Distributed Interactive Simulation (DIS), each user should render the visible part of the scene on-time and interact with other users in real-time.

Each user in DVE should load all the visible primitives in his Area of Interest (AoI) to give the impression of fluid motion. Most simulations load the entire scene into each distributed user from a CD or network at startup. However, a user only sees a small portion of the scene because most of the primitives in view frustum are occluded by the close visible primitives. Besides, every computer does not have enough capacity to load and run all the contents of the complex scenes. Especially mobile and handheld devices which have a limited capacity may not load all the contents. Hence, processing invisible primitives is useless and time consuming since it consumes the limited resources of computers such as memory and CPU power.

There are researches which focus on continuous and real-time delivery of the scene contents over network connections [39,40,41,43,44]. The purpose is to

send the PVS to a related user prior to participating in the virtual environment without a full download and update PVS as occasion may require. The main problem is that each user's visibility or AoI is different and should be computed individually. In an ideal manner, each user should only hold a subset of the scene corresponds to his AoI enabling to utilize his limited capacity and CPU power efficiently.

In addition to estimating AoI and sending the related PVS to a user, a user should also interact with the other users who he sees and who sees him in real-time. In most of the simulations, each user broadcasts his position and orientation updates to all users. Broadcasting all updates to everyone causes intensive network traffic when the number of user increases. Performance decreases and it will be a bottleneck in the end. There is no need for a user to communicate with the invisible users or the users out of his AoI. Therefore, it is enough for a user to communicate with the users in his AOI only.

In this study, we propose efficient visibility estimation for distributed virtual environments called Visibility-based Area of Interest (VbAoI) method. VbAoI method utilizes from region visibility method to compute PVS of static objects according to a region considering the occlusion in DVE. PVS for each navigable region is computed in preprocess conservatively using the ΔH and shrinking methods explained in Chapter 3 and Chapter 4. In the proposed VbAoI method, scene is organized as a graph and nodes of the graph map to the navigable regions where a distributed user may walkthrough on. Each distributed user only loads the PVS of static objects which are seen from the corresponding navigable region. A user renders the loaded PVS and interacts with the distributed users on the PVS of navigable regions in real-time.

The contributions of this study are summarized as follows:

1. The PVS of static objects are computed for each navigable region of urban environment in preprocess conservatively. Thus, each user loads the related PVS prior to entering the corresponding region.

2. All visible navigable regions from each navigable region are computed considering the occlusion of static objects in preprocess. A user should learn which navigable regions are visible from the region he is going to enter and which regions can see this region. Thus, a user only sends his updates to the users in these navigable regions and updates the positions and orientations of the other user simulators or avatars in PVS of regions dynamically.

3. VbAoI method abstracts users and environment using adjacency graph for navigable regions in DVE. Thus, a user does not need to be aware of the entire scenegraph of virtual environment. He is only aware of a subset of the graph which corresponds to his AOI in DVE.

4. VbAoI method builds a DVE infrastructure. After computing the PVS of static and dynamic objects conservatively, other optimization methods such as caching, prefetching, prioritization of content delivery, progressive loading, Level of Detail (LoD) and dead reckoning methods may be developed in this infrastructure as a future work.

The rest of this chapter is organized as follows: In Section 5.1, we present the details of proposed method. Section 5.2 discusses the results.

5.1 Visibility Based Area of Interest (VbAoI) Method

The proposed Visibility Based Area of Interest Method (VbAoI) is based on region-based visibility method explained in Chapter 4. VbAoI method estimates PVS of static primitives in preprocessing conservatively. The computed PVS may be splitted into active occluders such as buildings and passive occluders such as navigable regions which users may walkthrough on.

We modeled an urban environment as a testbed. The buildings and roads are modeled separately. A sampled urban environment is represented in Figure 5.1a. It is seen that roads are divided into navigable regions in Figure 5.1a. In

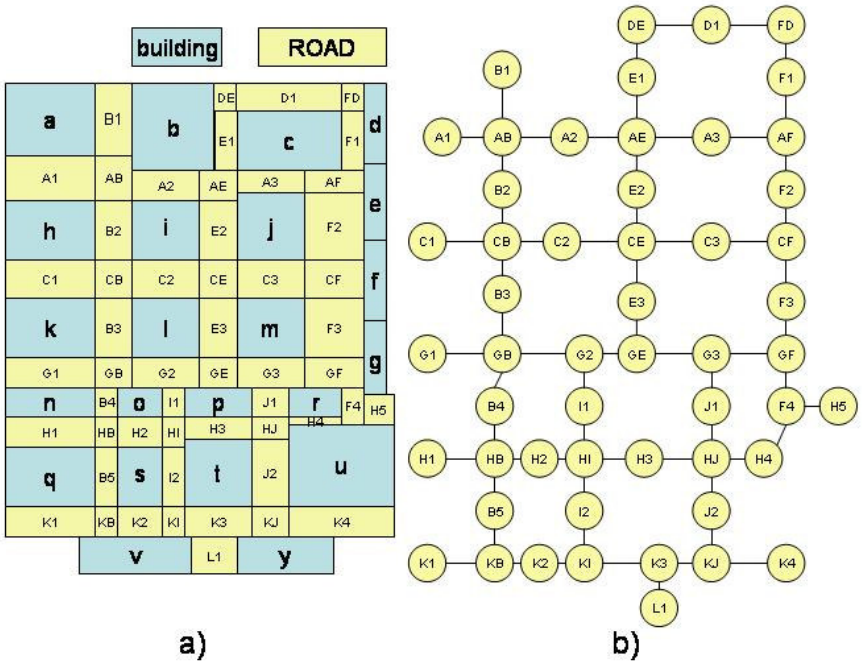


Figure 5.1: a) A part of urban. Buildings are colored blue. Roads are divided into navigable regions and colored yellow. b) An adjacency graph is constructed from the navigable regions. Each navigable region is represented as a node. Relation between nodes is represented by lines.

preprocessing, we initially build an adjacency graph from these navigable regions as seen in Figure 5.1b. Figure 5.1b shows which navigable regions (i.e., nodes or cell) are directly connected to each other. Adjacency graph abstract users from DVE. In other words, a user only knows the node he was inside and does not deal with whole DVE. From this point, the term of node, cell and region are used for the same meaning.

We compute PVS of buildings and navigable regions seen from each region considering the occlusion in DVE. We utilize ΔH and shrinking method together to compute PVS for each region. We determine sample points on the

boundary of each region as seen in Figure 4.2. Occluders (i.e., buildings) are shrunk ε-distance, which is the half distance of two consecutive sample points on the view cell's boundary as explained in Chapter 4. A smaller umbra is built by shrinking an occluder ε-distance. It guarantees that an occluded object does not visible from e-radius around the sample point as seen in Figure 4.1. We use ΔH method to build OH (i.e., fused umbra) for each sample point as explained in Chapter 3. A building is occluded for the whole view cell if it is contained in the fused umbra (OH) of the shrunk occluders for all sample points on the view cell's boundary as seen in Figure 4.2.

A unique id is assigned to each object in the environment. An object is sent to a client with its unique id. All the computed PVSs are associated with the corresponding nodes of adjacency graph. Occluders (buildings) and nodes in PVS are splitted into two sets of PVSo and PVSn. We use PVSo for PVS of occluders and PVSn for PVS of nodes.

In addition, there are channels associated to each node of the graph for data distribution management [54,55]. Channel id is the same id of the corresponding node. Channels are utilized for communications between the clients in DVE. Visibility in 360 degrees is mutual. In other words, if node A is visible from node B, node B is visible from node A. Thus, each client publishes his data from the channel id of his node and listens to the channel of nodes in Set of Neighbors (SNs). In the transmitted package between clients, channel ids are defined. A client only listens to and receives a packet from the subscribed channels.

The data structure of a node is given in Figure 5.2. The details of node items are summarized as follows:

1. *Node Id*: It is used to identify the corresponding node. It is also the channel id which a user in the corresponding cell publishes his position and orientation updates from.

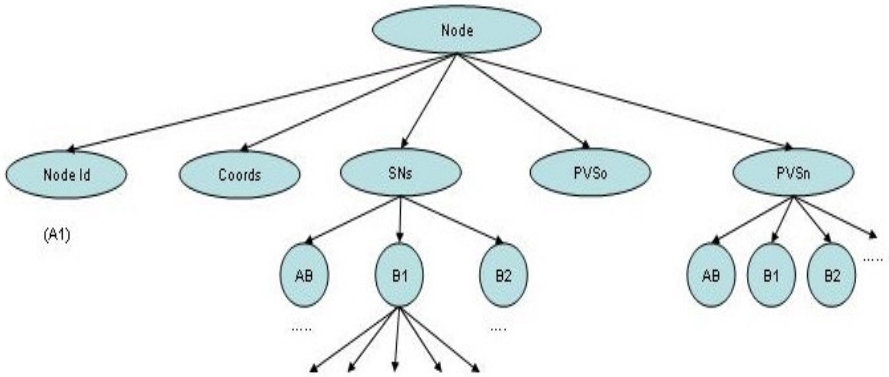


Figure 5.2: Data structure of a node

2. *Coordinates (Coords) of Node:* A user utilizes the coordinates of corresponding cell to check whether he is still in the cell.

3. *PVS of Occluders (PVSo)*: It is the PVS of visible static occluders (i.e., buildings). When a user is in a region, PVSo is enough to render all the visible static primitives.

4. *PVS of Nodes (PVSn):* It is the PVS of visible nodes from the node of user. It is also the set of channels which a user listens to.

5. *Set of Neighbor Nodes (SNs):* It is the set of neighbor nodes. When a node is sent to a user, a client or server transmits SNs with all their node data structure. It means that all the neighbor nodes are sent with their Node Ids, Coords, PVSo,PVSn.

The flowchart of VbAoI method is given in Figure 5.3. Top left dashed box summarizes the steps in the initial phase. A client participating DVE should follows these steps. Top right dashed box summarizes the steps performed by server because server has the entire graph. After client learned his node id and coordinates, the second communication may be performed with another client

in the corresponding cell or server. At the end of initial phase, a client has a subset of graph containing neighbor nodes.

Blue colored steps in the three dashed box below in Figure 5.3 are performed by the clients after finishing the initial phase. Each client performs these steps when he is in DVE. After all the data received, a client runs the following three threads simultaneously: 1) *Publish Thread:* A client moves in DVE according to the user input. When he moves, he always checks whether he is still in the cell. If he is still in the cell, he just only publishes his new positions and orientation to the channel of node. If not, he compares his position with the coordinates of neighbor nodes, discovers his new node, sends a leaving message to the channel of previous node, sends a welcome message including his position and orientation updates to the channel of new node and triggers subscribe thread. 2) *Subscribe Thread:* Subscribe or listen to the channels of PVSn and updates the position and orientation of potentially visible user simulators or avatars. If subscribe thread is triggered by publish thread, subscribe thread asks for the updates of SNs from a client/server who received welcome message. 3) *Render Thread:* It renders the PVSo and PVSn with respect to the client's position and orientation.

The flowchart in Figure 5.3 is represented as modules in Figure 5.4. Modules refer to simultaneously working threads in clients and server. We describe the steps of VbAoI methods in detail as follows: On the right hand side of Figure 5.4, main server loads adjacency graph at startup. After loading, server main loop runs. Network server thread periodically checks network if there is a new client connection request. When a client wants to participate in DVE, he sends a client connection request. In the first client connection request, a client only sends his starting position in DVE or accepts default starting position. Server replies this newly arrived client with his node id and Coords as seen in Figure 5.3. As a second step, client asks what he sees in DVE. If there is another client in the node which a new client wants to start, this client may reply the client connection request. He sends the requested data to the newly arrived client. If

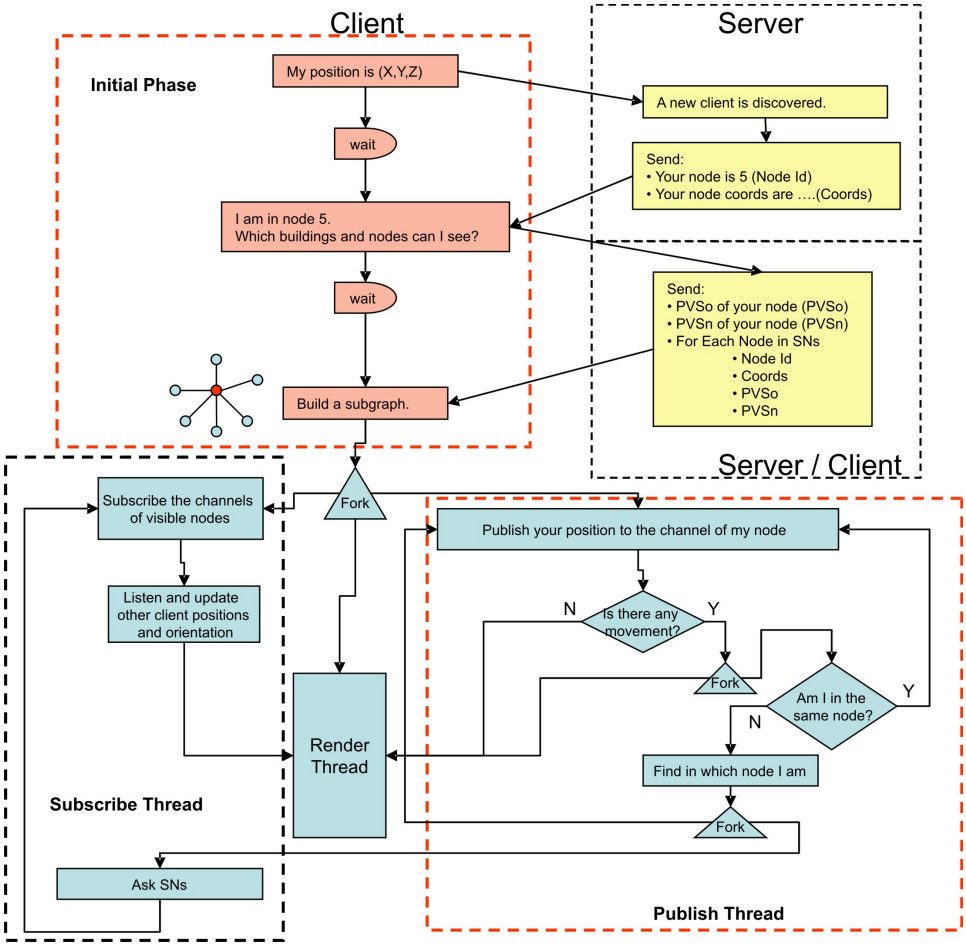


Figure 5.3: The flowchart of the proposed VbAoI method.

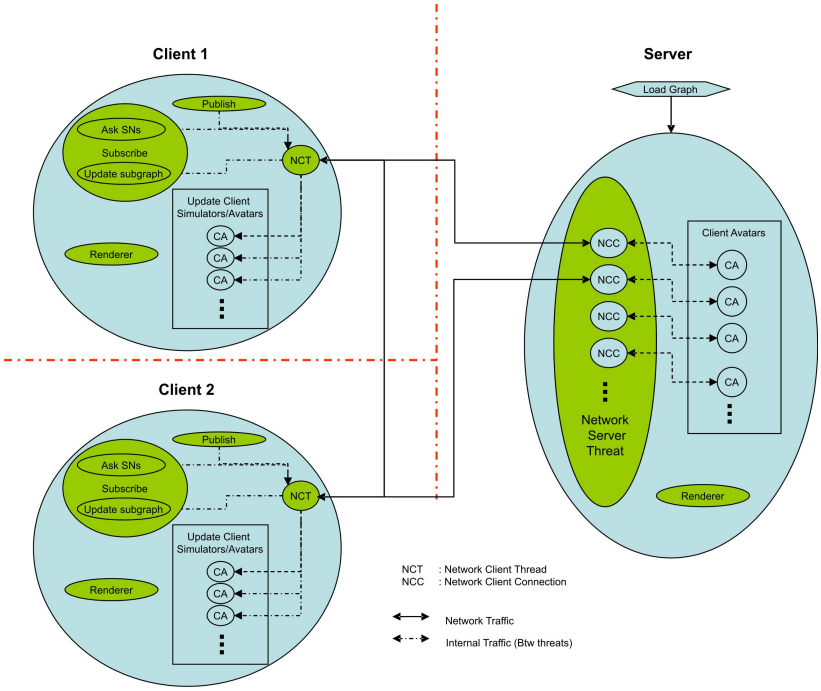


Figure 5.4: Modules of VbAoI method

not, server may reply the client request. The quickest client or server sends the requested data to the newly arrived client. The quickest client publishes data from the channel of the node. The other clients who received the first reply message do not send the requested data to newly arrived client. If the newly arrived client can not receive the requested data for a time period, he resends his request to the channel of his node.

Server also keeps the list of all users with node ids. Render thread of server does not run in normal mode and may be utilized if we want to render all the clients in DVE from bird's eye view as seen in Figure 5.5.

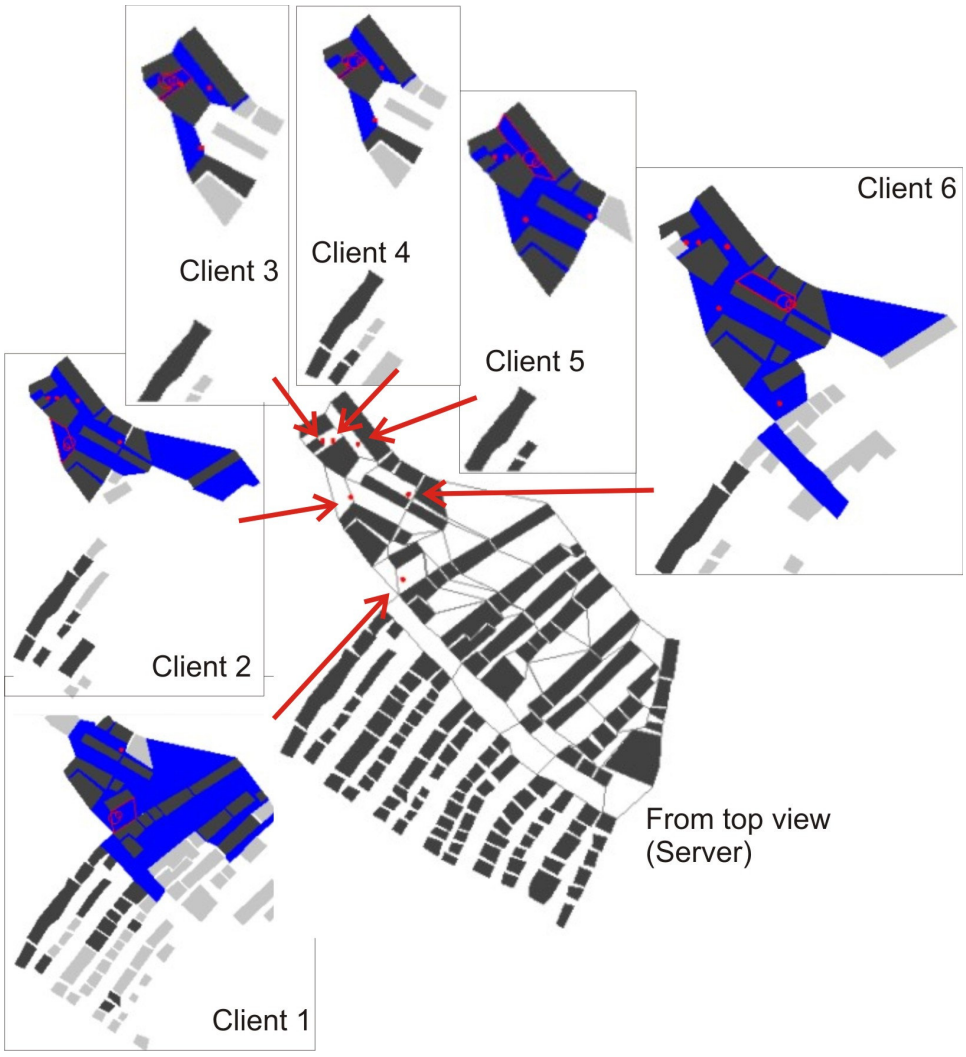


Figure 5.5: Screen shots from the testbed

Client modules are represented in the right hand side of Figure 5.4. A client participating in DVE has already received a part of graph, PVSo, and PVSn. Client knows the channel to publish his updates and the channels to be subscribed for listening to. If a client moves, he publishes his updates to the channel of the corresponding node. The clients subscribed to this channel receive updates and move user simulator or avatar. If a client passes to another node, he publishes his updates to channels of both nodes. Publishing his update to the channel of the node he left, informs the clients who listen to this channel about his leaving. Publishing his update to the channel of the node just entered is to inform the users who listen to this channel that he has just arrived to the node. When a client changes the node, he utilizes PVSs of the node he entered. At the same time, subscribe thread is triggered to update the part of adjacency graph, SNs, and their PVSn and PVSo. Node id, PVSn and PVSo of the entered node has already known by the client. Thus, user does not have flickering effect in DVE since he already has all the visible data. As seen in Figure 5.4, all communications of a client is performed by Network Client Thread.

5.2 Results and Discussion

We created a 3D model of an urban environment from the photographs taken from the satellite as a testbed. We modeled the roads and occluders separately as seen in the middle of Figure 5.5. We extend the buildings in z-axis randomly. There are 134 buildings and 93 navigable regions in the testbed as seen in Figure 5.5. We computed PVS of buildings and navigable regions seen from each region as explained in the previous section.

The snapshots taken from the testbed are gathered in Figure 5.5 to show how the proposed VbAoI method works. In the middle of Figure 5.5, the entire scene and the users in DVE are represented from top view in server. Buildings are colored dark gray and navigable regions are colored white and separated with line segments. Each screenshots of clients are taken from top to show their

positions in DVE. The visible navigable cells are colored blue in clients. The edges of client region are marked red. It is seen that Client 6 may see all the clients in the environment because his node may be visible from the nodes of other clients. Client 2 does not deal with Client 1 because the node of Client 2 can not see the node of Client 1. Client 1 only subscribes to the channel of Client 6 and updates its user simulator although Client 6 is subscribed to the channels of all clients to listen and update their user simulators.

The basic advantages of the proposed VbAoI method is to compute PVS of occluders and navigable regions in preprocess and model the environment as an adjacency graph. We build an adjacency graph of navigable regions for the whole scene as seen in Figure 5.1b. Adjacency graph enables to abstract the users from the environment. A user only knows the node which he is inside, his neighbor nodes and PVS of objects (i.e., buildings and roads) in his AoI. User does not deal with the whole DVE or an occluded object even if it is very close to user. DVE may be enlarged by adding new adjacency graphs to existing graph without users' notice. A client can use the extended part of the graph if that is in his AoI. Thus, modeling new towns or cities and joining them to DVE does not affect users in DVE. Furthermore, the number of servers in the environment may be increased and the graph of DVE may be divided into multiple servers. Thus, scalability of the DVE may be dynamically controlled with respect to the requirements of user, environment, network and etc.

A client does not need to communicate with a server if there is at least one user in the corresponding node or its neighbor nodes. A user which enters a new node may obtain all the required data from this user directly. Because each user has the data of its own node and neighbor nodes as seen in Figure 5.6a. A user in the environment does not care about server breakdown if there is at least one user in his node or neighbor nodes. If server gets available after a period, server will go on his service and update the user simulators listening to all channels.

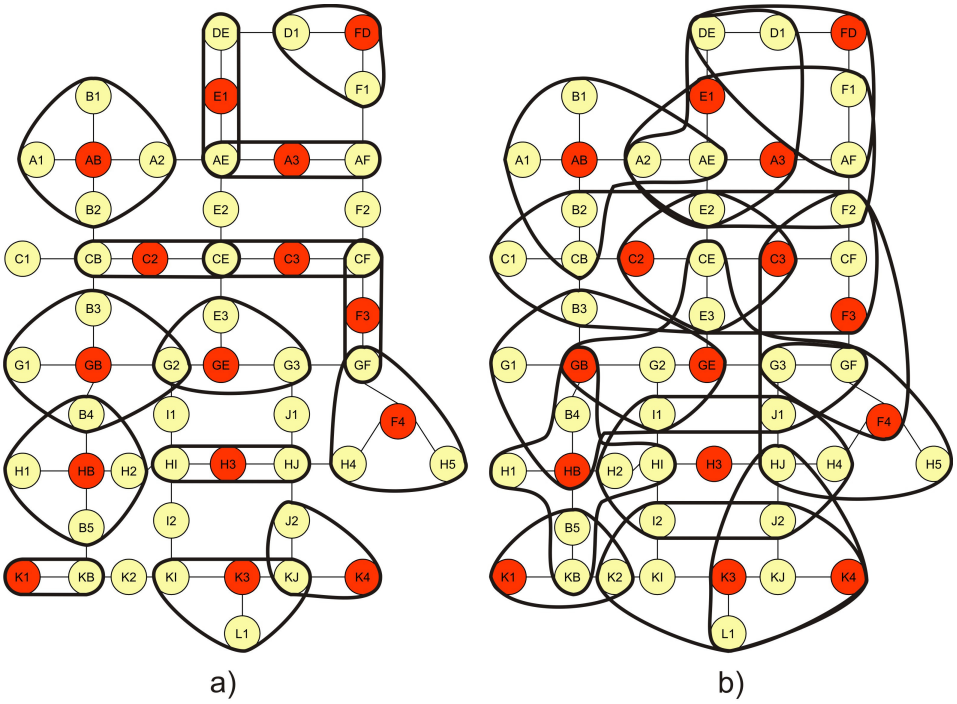


Figure 5.6: We assume that there are users in the red colored nodes. a) Default AoI of users covers the node of user and its neighbors. B) Extended AoI of users. User may extend his AoI if a user has enough bandwidth and capacity. There are more AoI overlaps in DVE. Much more parts of the graph are distributed to users and a user refers the server less frequently.

In the implementation, each user only obtains the data of his node and SNs as seen in Figure 5.6a. If a user has enough bandwidth and capacity, he may obtain more number of node data as seen in Figure 5.6b. Each user has a greater subset of graph which may overlaps more in DVE. Thus availability of the DVE is increased against the server breakdown by sharing the graph to clients more.

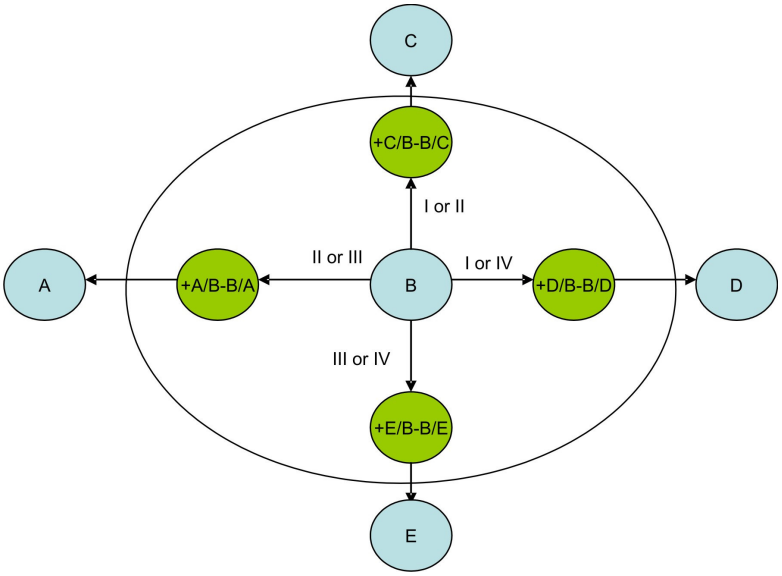


Figure 5.7: To avoid the data redundancy from sending the previously transmitted data, Differential data are sent to user prior to entering the new node.

It is seen that each of neighbor nodes of the graph keeps similar PVS because of occlusion in DVE. Visible primitives do not change much between the neighbor nodes. This gives the advantage of only sending the differential data when it is needed as seen in Figure 5.7. Thus, we may use the bandwidth effectively avoiding the data redundancy.

CHAPTER 6

CONCLUSION

The importance of visibility estimation in computer graphics increases parallel to the recent developments in computer technology and increased demand on interactive simulations and games. Current graphics hardware computes visibility in the last stage of rendering pipeline. However, every rendered primitive is not visible in the final image. Early culling of the invisible primitives in a complex scene is valuable for efficiency in the conventional rendering pipeline.

The motivation of this thesis is to develop a walkthrough application in distributed complex urban environment where buildings are the main source of occlusion. The viewer only sees a small part of the city because close buildings usually occlude the ones behind. The methods which are developed for 3D visibility can also be used for urban walkthroughs. However, the computation of full 3D occlusion includes a significant overhead since buildings are connected to the ground and may be modeled in 2½D.

In this thesis, we compute conservative visibility considering the occlusion for 2½D urban environments. We estimate visibility in the traversal stage of the conventional rendering pipeline to reduce the number of primitives that will be processed in the rest of the pipeline. We have developed the following three visibility methods based on occlusion culling throughout this thesis:

1. Incremental Occlusion Horizon Culling Method

The first method is a conservative incremental occlusion culling method called Delta-Horizon (ΔH). ΔH method is based on constructing OH, which is a set of connected lines passing just above all visible primitives, for culling the invisible primitives beyond. The proposed ΔH method, as in [20], computes the occlusion in 2½D, works on visibility based occluder selection, and supports both conservative visibility and occluder fusion. We utilize polar coordinates and build OH in object space although ΔH method may work in image space. The contributions of this method are summarized as follows:

- It computes PVS that are visible from the current viewpoint for each frame on the fly incrementally. Utilizing the coherence of occluders enables the incremental update of OH in consecutive frames.
- ΔH method allows flexible six DoF camera movements. Downs' method has only four DoF[20]. Solving this drawback of former method is going to increase the number of OH method implementation in walkthrough applications.
- It accommodates a 360-degree-wide field of view in one pass. This is desired for applications that need wide angle visibility such as radar simulations and panoramic viewing [73].
- PVS computed in object space may be utilized without recomputation when the camera zooming is required.

2. An Adjustable Occluder Shrinking Method

Adjustable occluder shrinking method computes PVS with respect to a region. Point-based visibility methods can not guarantee to compute PVS on time for complex scenes in real-time walkthrough applications. Therefore, we developed a new generic conservative occluder shrinking method for 2½D objects. Enhancing ΔH method with the proposed shrinking method enables to compute potential visible set once and utilize it in many consecutive frames. The contributions of this method are summarized as follows:

- It only shrinks the shadow frustum of a potential visible occluder and computes 30% more accurate PVS than former method [36OTEREF _Ref184658625 \h 36].

- It has no preprocess to shrink occluders and does not need to store the shrunk versions of occluders.

- It is more generic since all operations are performed in CPU without a need of specific graphics hardware.

- Occluders are selected regarding visibility from the view point rather than a selection method in preprocess.

- Shrink distance is dynamically adjustable with respect to the speed of viewer and frame rate on the fly. Hence, it enables to reach a compromise between PVS accuracy and real-time constraints.

3. Efficient Visibility for Distributed Virtual Urban Environments

Finally, we developed a visibility estimation method for distributed virtual urban environments called Visibility-based Area of Interest (VbAoI) method. A client in DVE does not need to load occluded parts of the scene and to communicate invisible clients who cause unnecessary communication overload in network. The scene is organized as a graph and each distributed client only be aware of the PVS of static objects and interacts with the PVS of dynamically grouped distributed clients. Nodes of the graph map to the navigable regions where a distributed client may walkthrough. The contributions of VbAoI method are summarized as follows:

- It estimates output sensitive PVSs for navigable regions in preprocessing. It assures the visible objects to be available at the distributed client on-time.

- It dynamically groups clients which may see each others in real-time. Thus, it enables these clients to communicate with each others and reduces the network traffic in DIS.
- It abstracts users and environment using adjacency graph for navigable regions in DVE. Thus, a user does not need to be aware of the entire scenegraph of virtual environment. He is only aware of a subset of the graph which corresponds to his AOI in DVE. VE may be extended dynamically without the notice of users in DVE. For example, you may model another city and add it to the existing DVE by only connecting the modeled city to a few node of the graph.
- It creates a DVE infrastructure. After computing the PVS of static and dynamic objects conservatively, other optimization methods such as caching, prefetching, prioritization of content delivery, progressive loading, Level of Detail (LoD) and dead reckoning methods may be developed in this infrastructure as a future work.

Computing conservative PVS and using adjacency graph for the navigable regions enables VbAoI method to create a DVE infrastructure. New optimization methods may be developed to increase the performance of the DVE as future work. The methods we foresaw may be more than the following list:

- Caching: Caching methods prevent to resend the previously sent object to client. Client may utilize his free memory as cache if it is available using a developed caching method for the needs of environment.
- Prefetching: The user in the environment has trends like moving forward for a while, following the same path when he goes to work and etc. Prefetching methods based on user trends may be adopted to increase the performance of the clients in the environment. For example, most of the nodes have one entrance and one exit with respect

to the movement direction. If a user enters the node from one side, he should exit from the other side. Thus, client may ask for the PVS of the following nodes without waiting. A new prefetching method based on probabilistic optimization is already implemented by Çevikbaş in [81].

- Prioritization, Progressive Loading and Simplification: In preprocessing, we have already computed conservative PVSs for all nodes. Computing PVS in preprocessing gives the advantage of prioritization of objects in the environment. In addition, we may model the simplified versions of the objects using Level of Detail (LoD) method of progressive meshes. Furthermore, we may load the PVS to the client progressively.

- Dead reckoning: VbAoI method computes the PVS of navigable nodes for each node in preprocessing. Before a user enters the node, he has the list of visible nodes. Thus we may prioritize the visible nodes with respect to its distance and direction. A client may use dead reckoning methods for the users of the nodes out of view frustum or the nodes away from the client.

REFERENCES

[1] Cohen-Or D., Chrysanthou Y., Silva C., Durand F.: A Survey of Visibility for Walkthrough Applications. IEEE Trans. on Visualization and Computer Graphics, 9, 3, (2003), 412-431.

[2] Wonka P.: Occlusion Culling for Real-time Rendering of Urban Environments. PhD Thesis, Technische Universitat Wien, 2001.

[3] Watt A.:3D Computer Graphics. Second Edition,Addison-Wesley,England,1996.

[4] Es.Ş.A.: Accelerated Ray Tracing Using Programmable Graphics Pipeline. PhD Thesis, Middle East Technical University, Ankara, Jan 2008.

[5] Airey J.: Increasing Update Rates in the Building Walkthrough System with Automatic Model-space Subdivision and Potentially Visible Set Calculations. PhD Thesis, University of North Carolina, Chapel Hill, 1991.

[6] Bittner J.: Hierarchical Techniques for Visibility Computations. PhD Dissertation, Czech Technical University, Prague, Dec. 2002.

[7] Bittner J. and Wonka P.: Visibility in Computer Graphics. Journal of Environment and Planning B: Planning and Design, 30, 5, (Sept 2003), 725-729.

[8] Greene N.: Occlusion Culling with Optimized Hierachical Z-Buffering. *ACM SIGGRAPH Course Notes #30*, 2001.

[9] Koldas G., Isler V., Lau R.W.H.: Six Degrees of Freedom Incremental Occlusion Horizon Culling Method for Urban Environments. Advances in Visual Computing (Proc. 3rd International Symp.,ISVC 2007) Springer, Lake Tahoe, NV, USA, Nov 2007 LNCS 4841, 792-803.

[10] Durand F.: 3D Visibility: Analytical Study and Applications, PhD Thesis, University of Grenoble I-Joseph Fourier,Jul. 1999.

[11] Coorg S. and Teller S.: Temporally Coherent Conservative Visibility. Proc. ACM Symp. Computational Geometry, (1996), 78-87.

[12] Coorg S. and Teller S.: Real-time Occlusion Culling for Models with Large Occluders. Proc. ACM Symp. on Interactive 3D Graphics, (1997) 83-90.

[13] Hudson T., Manocha D., Cohen J., Lin M., Hoff K., Zhang H.: Accelerated Occlusion Culling Using Shadow Frusta. Proc. ACM Symp. Computational Geometry, 1-10, 1997.

[14] Bittner J., Havran V., Slavik P.: Hierarchical Visibility Culling with Occlusion Trees. Proc. Computer Graphics International '98, 207-219.

[15] Chin N. and Feiner S.: Near Real-time Shadow Generation Using BSP Trees. ACM Computer Graphics, 23,3 (1989), 99-106.

[16] Gummerus S.: Conservative from Point Visibility. Master's Thesis, University of Tampere, 2003.

[17] Greene N., Kass M., Miller G.: Hierarchical Z-buffer Visibility. Proc. ACM SIGGRAPH, (1993), 231–240.

[18] Zhang H.: Effective Occlusion Culling for the Interactive Display of Arbitrary Models. Ph.D. Thesis, University of North Carolina, Chapel Hill, July 1998.

[19] Wonka P. and Schmalstieg D.: Occluder Shadow for Fast Walkthroughs of Urban Environments," Computer Graphics Forum, 18,3, (1999) 51-60.

[20] Downs L., Moeller T., Sequin C.: Occlusion Horizons for Driving Through Urban Scenery. Proc. ACM Symp. on Interactive 3D Graphics, (2001) 121-124.

[21] Lloyd B. and Egbert P.: Horizon Occlusion Culling for Real-time Rendering of Hierarchical Terrains. Proc. Conf. on Visualization, (2002), 403-410.

[22] Scott N., Olsen D., Gannet E.: An Overview of the Visualize Fx Graphics Accelerator Hardware. The Hewlett-Packard Journal, (May 1998), 28-34.

[23] Severson K.: Visualize Workstation Graphics for Windows NT. HP Product Literature, 1999.

[24] Klosowski J., Held M., Mitchell J., Sowizral H., Zikan K.: Efficient Collision Detection Using Bounding Volume Hierarchies of K-dops. IEEE Trans. on Visualization and Computer Graphics. 4,1, (Jan./Mar. 1998), 21-36.

[25] Morein S.: ATI Radeon HyperZ Technology. Proc. SIGGRAPH/Eurographics Graphics Hardware Workshop, Hot3D Session, 2000.

[26] Nvidia Lightspeed Memory Architecture II. Technical brief, NVIDIA, Jan. 2002.

[27] DirectX 9 SDK. Microsoft Corporation, 2003. 14 February 2008. http://msdn2.microsoft.com/tr-tr/directx/default(en-us).aspx

[28] Segal M. and Akaley K.: The OpenGL Graphics System. A specification (Version 2.0), Silicon Graphics Inc., Oct., 2004.

[29] Airey J.: Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations. PhD thesis, University of North Carolina, Chappel Hill, 1991.

[30] Airey.J.M., Rohlf J. H., Brooks F. P.Jr. : Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. Computer Graphics (1990 Symposium on Interactive 3D Graphics), March 1990, 24(2):41–50.

[31] Teller S.: Visibility Computations in Densely Occluded Environments. PhD thesis, University of California, Berkeley, 1992.

[32] Schaufler G., Dorsey J., Decoret X., Sillion F.X..:Conservative Volumetric Visibility with Occluder Fusion. Proc. SIGGRAPH 2000, July 2000, 229–238.

[33] Durand F., Drettakis G., Thollot J., Puech C.. Conservative Visibility Preprocessing Using Extended projections. Proc. of SIGGRAPH 2000, July 2000, 239–248.

[34] Koltun V., Chrysanthou Y., Cohen-Or D.,. Hardware-accelerated from-Region Visibility Using a Dual Ray Space. Rendering Techniques 2001: 12th Eurographics Workshop on Rendering, Eurographics, June 2001, 205–216.

[35] Wonka P., Wimmer M., Schmalstieg D.: Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs, Rendering Techniques 2000 (Proc. Eurographics Workshop on Rendering 2000), Eurographics, Springer Verlag Wien New York, June 2000, 71-82.

[36] Wonka P., Wimmer M., Sillion F.: Instant Visibility. Computer Graphics Forum (Proc.of Eurographics 2001), 20, 3, Sept. 2001.

[37] Quake. 14 February 2008. http://www.idsoftware.com/

[38] Diablo II. 14 February 2008. http://www.blizzard.com/diablo2/

[39] Fujimoto R.M.: Parallel and Distributed Simulation Systems, John Wiley & Sons, Inc.,USA, 2000.

[40] Hu S.Y., Chang S.C., Sung W.L., Jiang J.R.: A Case for 3D Streaming on Peer-to-Peer Networks. Proc.of 11the Intl.Conf. on 3D Web Technology (Web3D 2006), 2006.

[41] Hu S.Y., Chang S.C., Sung W.L., Jiang J.R.: A Case for Peer-to-Peer 3D Streaming, Ascend Technical Report: ASCEND-TR-002-2006A

[42] Electronic Arts.14 February 2008. http://www.ea.com

[43] Singhal S., Zyda M.: Networked Virtual Environments, ACM Press, 1998.

[44] Li F.W.B, Lau R.W.H. Kilis D. , GameOD: An Internet Based Game-On-Demand Framework, *Proc. of ACM VRST*, pp. 129-136, Nov. 2004.

[45] Hagsand O.: Interactive Multiuser VEs in the DIVE System. *IEEE Multimedia*, **3**(1):30–39, 1996.

[46] Leigh J., Johnson A., Vasilakis C., DeFanti T.: Multi-perspective Collaborative Design in Persistent Networked Virtual Environments. *Proc. IEEE VRAIS*, pp. 253–260, 1996.

[47] Waters R., Anderson D. Barrus J. Brogan D. et.al.: Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability. *Presence*, **6**(4):461–480, Aug. 1997.

[48] Saar K.: VIRTUS: A Collaborative Multi-User Platform. *Proc.VRML*, pp. 141–152, 1999.

[49] Goggle Earth. 14 February 2008. http://earth.google.com/

[50] Falby J., Zyda M., Pratt D., Mackey R.: NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers & Graphics*, **17**(1):65–69, 1993.

[51] Macedonia M., Zyda M., Pratt D., Brutzman D., Barham P.: Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments. *Proc. IEEE VRAIS*, pp. 38–45, 1995.

[52] Greenhalgh C., Benford S.: MASSIVE: A Distributed Virtual Reality System Incorporating Spatial Trading. *Proc. ICDCS*, pp. 27–34, 1995.

[53] Das T., Singh G., Mitchell A., Kumar P., McGhee K.: NetEffect: A Network Architecture for Large-scale Multi-user Virtual World. *Proc. ACM VRST*, pp. 157–163, 1997.

[54] Ivan Chang Kok Ping. Hla Performance Measurement, MS Thesis, Naval Postgraduate School, Monterey, California, 2000.

[55] Defense Modeling and Simulation Office, "HLA Interface Specification," Version 1.3.

[56] Deb S., Narayanan P. J.: Design of a Geometry Streaming System. *Proc. ICVGIP*, 2004, pp. 296–301.

[57] Teler E., Lischinski D.: Streaming of Complex 3D Scenes for Remote Walkthroughs," *EUROGRAPHICS*, vol. 20, no. 3, 2001.

[58] Rusinkiewicz S., Levoy M.: Streaming Qsplat: A Viewer for Networked Visualization of Large, Dense Models," *Proc. ACM I3D*, 2001, pp. 63–69.

[59] Hoppe H.: Progressive Meshes. *Proc. ACM SIGGRAPH*, 1996, pp. 99–108.

[60] Kim J., Lee S., Kobbelt L.: View-dependent Streaming of Progressive Meshes. *Proc. SMI'04*, 2004, pp. 209–220.

[61] Pajarola R., Rossignac J.: Compressed Progressive Meshes. *IEEE Trans. Vis. Comput. Graph.*, vol. 6, no. 1, pp. 79–93, 2000.

[62] Chen Z., Bodenheimer B., Barnes J. F.: Robust Transmission of 3D Geometry Over Lossy Networks. *Proc. ACM Web3D*, 2003, pp. 161–ff.

[63] Yang S.,. Kuo C.-C. J.: Robust Graphics Streaming in Walkthrough Virtual Environments via Wireless Channels. *Proc. Globecom*, 2003.

[64] Chen and B.-Y., Nishita T.: Multiresolution Streaming Mesh with Shape Preserving and QoS-like Controlling. *Proc. Web3D*, 2002, pp. 35–42.

[65] Schmalstieg D., Gervautz M.: Demand-driven Geometry Transmission for Distributed Virtual Environments. *Computer Graphics Forum*, vol. 15, no. 3, pp. 421–433, 1996.

[66] Hesina G. and Schmalstieg, D.: A Network Architecture for Remote Rendering. *Proc. Intl. Wksp. DIS-RT*, 1998, p. 88.

[67] Chim J., Lau R. W. H., Leong H. V., Si A.: Cyberwalk: A Webbased Distributed Virtual Walkthrough Environment. *IEEE Trans. on Multimedia*, vol. 5, no. 4, pp. 503–515, 2003.

[68] Brown B., Bell M.: Cscw at Play: 'There' as A Collaborative Virtual Environment. *Proc. ACM CSCW*, 2004, pp. 350–359.

[69] Rosedale P., Ondrejka C.: Enabling Player-Created Online Worlds with Grid Computing and Streaming. Gamasutra Resource Guide, 2003. 14 February 2008. http://www.gamasutra.com/resource_guide/20030916 /rosedale_pfv.htm

[70] Olbrich S., Pralle H.: Virtual Reality Movies - Real-time Streaming of 3D Objects. *Computer Networks*, vol. 31, no. 21, pp. 2215–2225, 1999.

[71] Gerndt A., Hentschel B., Wolter M., Kuhlen T., Bischof C.: Viracocha: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments. *Proc. SC2004*, 2004.

[72] Cheng L., Bhushan A., Pajarola R., Zarki M. E.: Real-time 3D Graphics Streaming Using Mpeg-4," in *Proc. IEEE/ACM Wksp. on Broadband Wireless Services and Appl.*, 2004.

[73] Koldas G., Cevikbas S.B., Isler V.: Echo Generation for Mobile Radar Simulation. Proc. 2nd IEEE Int'l. Conf. on Tech. for Homeland Security and Safety, (2006), 79-87.

[74] VRSG Radar. 14 February 2008 http://www.metavr.com/products/vrsg /radar.html

[75] Worsham R.H., Northrop Grumman radar simulation (AVSIM), Proceedings of the IEEE Radar Conference, 176-186 (2002).

[76] Radar: Modular Ground Based. 14 February 2008. http://www.edocorp.com/RadarGround.htm

[77] James W. J. III: A Three-Dimensional Ray Tracing Simulation of a Synthetic Aperture Ground Penetrating Radar System. MS Thesis, Rochester Institute of Technology, Rochester, New York (2002).

[78] Savides T. and Dwolatzky B.: Radar Simulation Using the Shooting and Bouncing Ray Technique. IEEE CCECE 2003 Canadian Conference on Electrical and Computer Engineering, 307-310 (2003).

[79] Andersh D., Moore J., Kosanovich S., Kapp D. Bhalla R., Kipp R., Courtney T. Nolan A., German F. and Cook J. Xpatch4: The Next Generation in High Frequency Elektromangnetic Modeling and Simulation Software. IEEE International Radar Conference (2000).

[80] Yang C., Wu B. and Ko C.: A Ray-tracing Method for Modeling Indoorwave Propagation and Penetration. IEEE Trans.Ant.Prop.,46, 907-919 (1998).

[81] Çevikbaş.Ş.B.: Visibility Based Prefetching with Probabilistic Optimization. MS Thesis, Middle East Technical University, Ankara, Jan. 2008.

VITA

PERSONAL INFORMATION

Surname, Name	: KOLDAŞ, Gürkan
Nationality	: Turkish (TC)
Date and Place of Birth	: Nov 29, 1970, İstanbul
Marital Status	: Married
Phone	: +90 312 4033415/535 3177450
Email	: gurkan@dzkk.tsk.mil.tr

EDUCATION

Degree	**Institution**	**Year of Graduation**
MS	METU, Computer Engineering	2000
BS	Naval Academy, E.E.Eng.	1993
High School	Naval High School	1989

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2000-Present	Naval HQ	Sys.Adm.
1996 August	Deniz Eğt.ve Öğr.K.lığı	Planning Officer
1995 August	Konca Ana Mayın Gr.K.lığı	HQ Officer
1994 March	TCG Yücetepe	Weapon Eln.Sys.Officer

FOREIGN LANGUAGES

English

PUBLICATIONS

1. Koldas G., Isler V., Lau R.W.H.: Six Degrees of Freedom Incremental Occlusion Horizon Culling Method for Urban Environments. Advances in Visual Computing (Proc. 3rd International Symp.,ISVC 2007) Springer, Lake Tahoe, NV, USA, Nov 2007 LNCS 4841, 792-803.
2. Koldaş G. Çevikbaş Ş.B., İşler V.: Echo Generation for Mobile Radar Simulation.The Second IEEE Int. Conf. on Technologies for Homeland Security and Safety (TEHOSS 2006), 9-13 October 2006, İstanbul, TURKEY.
3. Koldaş G. Çevikbaş Ş.B., İşler V.: Dağıtık Simulasyon Uygulamalarında Gerçek Zamanlı Görselleştirme : Ufuk Tabanlı Görünürlük Tespit Yöntemi. 3. Savunma Teknolojileri Kongresi (Savtek 2006), 29-30 Haziran 2006, ODTÜ, Ankara, TURKEY (in TURKISH).
4. Koldas G., Isler V.: Multiresolution Behavioral Modeling in a Virtual Environment. 33rd Annual Simulation Symp.- Advanced Simulation Technologies Conf. 2000 (ASTS 2000), 16-20 Nisan 2000, Washington, USA.
5. Koldas G., Isler V.: Dynamic Simulation of a Sailing Boat. 14th Int. Symp. on Computer and Information Sciences 1999 (ISCIS 1999), Ekim 1999, Kuşadası, İZMİR, TURKEY.
6. Koldas G., Isler V.: Askeri Uygulamalarda Üç Boyutlu Hızlı Animasyon Teknikleri. 2000'li Yıllarda Uzay Havacılık ve Savunma Teknolojilerinin Öncelikleri Sempozyumu, 29-30 Nisan 1999 Hava Harp Okulu, İstanbul, TURKEY (in TURKISH).