Ş.B.ÇEVİKBAŞ

METU

2008

VISIBILITY BASED PREFETCHING WITH SIMULATED ANNEALING

ŞAFAK BURAK ÇEVİKBAŞ

JANUARY 2008

VISIBILITY BASED PREFETCHING WITH SIMULATED ANNEALING

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

BY

ŞAFAK BURAK ÇEVİKBAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2008

Approval of the thesis

VISIBILITY BASED PREFETCHING WITH SIMULATED ANNEALING

submitted by **Şafak Burak ÇEVİKBAŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering** , Middle East Technical University, by,

Prof. Dr. Zafer Dursunkaya _____
Dean, **Graduate School of Natural and Applied Sciences**

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

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

Examining Committee Members:

Prof. Dr. İsmail Hakkı Toroslu _____
Computer Engineering, **METU**

Assoc. Prof. Dr. Veysi Isler _____
Computer Engineering, **METU**

Assoc. Prof. Dr. Uğur Güdükbay _____
Computer Engineering, **Bilkent**

Dr. Cevat Şener _____
Computer Engineering, **METU**

Asst. Prof. Dr. Tolga Can _____
Computer Engineering, **METU**

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.

Şafak Burak Çevikbaş

ABSTRACT

VISIBILITY BASED PREFETCHING WITH SIMULATED ANNEALING

Şafak Burak ÇEVİKBAŞ
M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Veysi Isler

January 2008, 69 pages

Complex urban scene rendering is not feasible without culling invisible geometry before the rendering actually takes place. Visibility culling can be performed on predefined regions of scene where for each region a potential visible set of scene geometry is computed. Rendering cost is reduced since instead of a bigger set only a single PVS which is associated with the region of the viewer is rendered. However, when the viewer leaves a region and enters one of its neighbors, disposing currently loaded PVS and loading the new PVS causes stalls. Prefetching policies are utilized to overcome stalls by loading PVS of a region before the viewer enters it. This study presents a prefetching method for interactive urban walkthroughs. Regions and transitions among them are represented as a graph where the regions are the nodes and transitions are the edges. Groups of nodes are formed according to statistical data of transitions and used as the prefetching policy. Some heuristics for constructing groups of nodes are developed and Simulated Annealing is utilized for constructing optimized groups based on developed heuristics. The proposed method and underlying application of Simulated Annealing are customized for minimizing average transition cost.

Keywords: Visibility, Prefetching, Simulated Annealing, Urban, Walkthrough

ÖZ

GÖRÜNÜRLÜK TABANLI OLASILIKSAL ERKEN OKUMA

Şafak Burak ÇEVİKBAŞ

Yüksek Lisans, Fen Bilimleri Bölümü

Tez Yöneticisi: Doç. Dr. Veysi Isler

Ocak 2008, 69 sayfa

Karmaşık şehir ortamları, herhangi bir ayıklama kullanılmadan gerçeklenebilir değillerdir. Ayıklama yapmak için kullanılan bir yöntem, sahneyi hücrelere bölmek ve her hücreye bir görünebilir nesneler kümesi atamaktır. Sadece izleyicinin bulunduğu hücrenin görünebilir nesne kümesinin gerçeklenmesi, bütün sahne ya da sahnenin büyük bir kısmının gerçeklenmesine göre çok daha az kaynak gerektirir. Buna rağmen akıcı bir gerçekleme için hücre geçişlerindeki yavaşlamanın üstesinden gelinmelidir. Bu sorunu aşmak için kullanılan yaklaşıma erken yükleme adı verilir. Erken yükleme ile gerçekleme için gerekli veri daha izleyici hücreye girmeden önce yüklenir. Bu çalışma şehir ortamında etkileşimli gezinti uygulamaları için kullanılabilecek bir erken yükleme yöntemi sunar. Yöntem sahneyi bir çizge olarak ifade eder ve bu çizge üzerinde düğüm grupları tanımlar. Tanımlanan gruplar erken yükleme için girdi olarak kullanılır. Grupların oluşturulması için Benzetimli Tavlama algoritması ve buluşsallar. Yöntem ortalama geçiş masrafının azaltılması ya da en yüksek geçiş masrafının azaltılması gibi farklı amaçlar için özelleştirilebilir.

Anahtar Kelimeler: Görünürlük, Ön-okuma, Şehir, Benzetimli Tavlama

To Beautiful People of My Life.

ACKNOWLEDGEMENTS

The author of this study presents his gratefulness to:

… his supervisor Assoc.Prof. Dr. Veysi İŞLER for innovative guidance, criticism and encouragement;

… his pioneer Gürkan KOLDAŞ for his leading and support;

… his parents Nazmi and Nurten ÇEVİKBAŞ for their support, motivation, and patience;

… his friend Elif TÜFEKÇİ for her friendship and bolèos;

… his friends Cihan KÜÇÜKGÖZE and Fatih Ertan ARIN for remembering his name after the long time they have not seen him while he was busy with this study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Interactive frame-rates are required for walkthrough applications. Even the best performing hardware can become insufficient with the increasing size of the scene data. Rendering cost can be reduced by culling the invisible geometry before the actual rendering. Visibility studies concentrate on identifying then culling invisible away before they are sent to rendering pipeline.

Simple method of culling is frustum culling where the scene geometry is tested against the view frustum and those staying out of the frustum are culled. However, this method is far from being sufficient. After frustum culling there is still significant amount of invisible geometry that can be culled. These are the away facing surfaces and geometry occluded by other geometry. Surfaces facing away from the viewer do not contribute to rendered image, so they can be culled away and this process is called Hidden Surface Removal. Many improvements on Hidden Surface Removal are provided by researchers.

Other group of geometry to cull besides Hidden Surfaces is the occluded geometry. In complex scenery occlusion between scene geometry is inevitable. Studies focus on exploiting this fact. Traditional approach of visibility aims computing exact visible set. However computational cost is very high and consequently it is not feasible for interactive applications. Methods with lower computational costs were developed based on the concept of conservative visibility where the visible set is overestimated for the sake of computational cost. In other words a trade-off between culling accuracy and culling speed is

provided. Overestimated set of visible geometry is referred as Potential Visible Set (PVS).

Visibility studies cluster around two approaches point-based and region/cell based. While point-base methods recalculate PVS for each location of the viewer at run-time, region based methods calculate PVS usually at preprocessing. Region based methods separate the scene into view-cells. The geometry visible from each cell is computed and attached as PVSs of cells. At run-time, PVS of the view-cell containing the viewer is sent to rendering pipeline. Region based methods provide infrastructure necessary for effective prefetching. With prefetching, next move of the viewer is predicted before it happens and necessary PVS is loaded. Starting from this point cell and region are used interchangeably.

The motivation of this study is to develop a method for improving prefetching policies used for cell based rendering of complex urban scenery where the occluders and occludees are buildings and view-cells built from spaces between the buildings. In dense city scenery, most of the buildings are occluded by the others and only a small portion of the scene can be seen by the viewer. The urban scene can be represented by a function of height, in other words it is in fact 2.5D, not 3D with hanging objects. This property of the urban scene is exploited by the visibility methods to decrease computation costs.

Basic prefetching policy that can be applied to region based rendering of urban environment is fetching PVSs of all neighbor cells of the current cell. For a grid representation of scene, this means fetching cell PVSs in groups of nine. Instead fetching subsets of neighbors according to viewer tendency of motion will be an improvement to this approach.

This study presents adaptation of a generic probabilistic optimization algorithm to prefetching. In fact, the method does more than optimization and provides a general approach for developing prefetching policies. The method converts prefetching policy development to defining heuristics which exploits scene semantics and scene structure.

The scene to be rendered is abstracted to a graph where nodes are the view-cells and edges are the transitions among view-cells. Each transition is assigned a cost generated by the delta loading of destination cell's PVS. Delta loading means loading data which is not already in the memory. In other words, delta loading is omitting preexisting data from fetching.

Using the graph, prefetching is mapped to a graph partitioning problem. Groups of node are identified and sets of buildings called Prefetching Set assigned to the groups. When the user is in a cell and completed loading of PVS, the idle IO time is utilized to load PS of groups containing the current cell. With this mapping graph partitions are used to express data to be prefetched. A standard way of finding the best partitioning is not available because the concerns for prefetching may be different and exact solution may never exist. Therefore Heuristics are defined for group forming and PS construction. These heuristics are accompanied by evaluation schemas to find best solution on the graph. Simulated Annealing is utilized for optimizing the solution, in other words; finding the nearest approximation to best partitioning.

Proposed method provides a flexible and extendable approach for developing prefetching policies. One of the major contributions is mapping prefetching to an abstract problem of graph partitioning. The other is adapting a generic

optimization algorithm for solution finding therefore allowing development of heuristics. Sample implementation of the method proposes a prefetching policy which considers both scene properties and structure. The method favors high probability transitions and central view-cells during construction of cell groups, therefore considers scene geometry and view-cell characteristics. The method is customized for optimization of popular paths for limited resource environment, average case performance, and worst case performance.

The rest of the paper is organized as follows. Chapter II reviews related work on visibility methods and prefetching for region based methods. Chapter III explains the method in detail. Chapter IV presents implementation details, Chapter V discusses the results. Finally, Chapter VI presents the conclusion and future work.

CHAPTER 2

RELATED WORK

Increasing complexity of models and the need for interactive walkthroughs in them, led visibility studies to become a significant area. Starting from the early stages of computer graphics area, visibility has always been a fundamental problem. Algorithms are designed to determine which parts of the scene is visible on the scene and which part was invisible. Primitive approach is hidden line removal on vector displays. Later this approach is replaced with hidden surface removal techniques after the emergence of raster displays.

Required framerate for an interactive walkthrough application is at least 20 Hz [1]. There always have been scenes more complex than the best performing hardware can handle with brute force. Visibility algorithm has to be utilized for necessary acceleration of complex scene rendering. Since the early days of visibility computation, many approaches including the visibility culling methods have been built.

Visibility culling methods filter invisible geometry out before it is sent to rendering pipeline. Visibility culling achieves great acceleration on scenes crowded by geometries occluding each other. Examples are indoor and outdoor walkthrough applications. Geometry to be culled can be grouped into three as shown in Figure 1. Visibility culling:

1. **View-frustum culling:** Those stay out of the view-frustum.

2. **Back-Face culling:** Surfaces facing away from the viewer.
3. **Occlusion Culling:** Geometry occluded by other geometry in the scene.

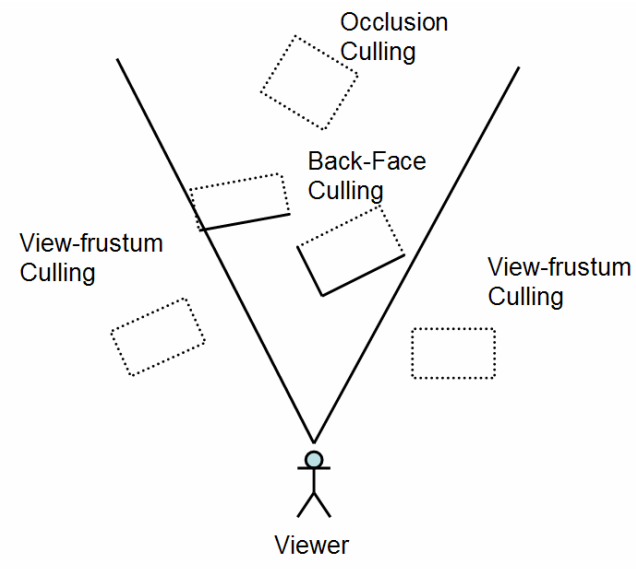

Figure 1. Visibility culling.

Since processing of each polygon is too expensive, algorithms generally work on hierarchies of scene geometry where lower levels on the hierarchy present a smaller portion of the scene. An important concept necessary to differentiate algorithms is conservative visibility [2, 3]. While computing the Potential Visible Set (PVS) of scene objects, algorithms which include all visible objects from the reference point or region are called conservative algorithms. Usually for the sake of computation cost, algorithms overestimate the PVS. In other words to decrease computation costs as a trade off, PVS usually include all visible objects of the scene with a small set of invisible objects which increase

the rendering cost [2, 3]. Methods which are not considered as conservative methods, risk culling visible geometry for the sake of rendering performance.

One of the available classifications has to be highlighted: online visibility culling and offline visibility culling. While the former determines visibility of scene geometry on the fly at run-time, the latter precompute visibility at preprocessing stage [4].

2.1 ONLINE VISIBILITY CULLING ALGORITHMS

Online visibility culling algorithms recompute PVS at each change of viewer location, i.e. the viewpoint. Methods computing the PVS for each change of viewpoint are classified as point-based or from-point visibility culling methods [4]. Some algorithms amortize the cost of PVS computation by doing the computation for a small neighborhood of the viewpoint. Once the PVS is computed, it is valid for several frames. This amortized time is allocated for computation of consequent PVS.

Object precision from-point visibility culling methods exploit relations between objects of the scene. They are built on the existence of large occluders. Coorg and Teller [5, 6] proposed a method which uses a selection of large convex occluders to compute the occlusion in the scene. Two objects are considered, one as occluder the other as the ocludee. Supporting and separating lines are defined based on selected couple of objects as in Figure 2. Supporting and Separating Lines They partition the space cells from where the view is invariant. This means, visible set is constant until the viewer passes one of these

lines. In 3D lines are replaced with planes and point generating these lines are replaced with edges of the objects. However the complexity of the method is quite high. Figure 3. Cells generated by three polygons. illustrates exponential complexity of the method.

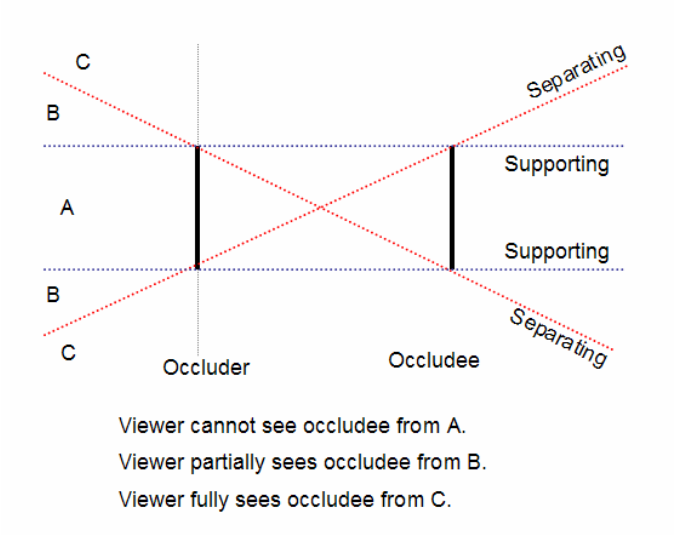

Figure 2. Supporting and Separating Lines.

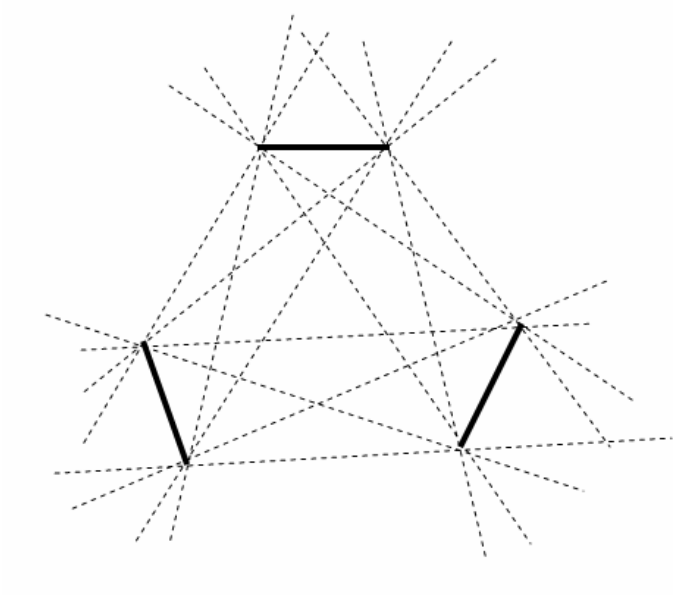

Figure 3. Cells generated by three polygons.

Another similar way of utilizing occlusion relations is proposed by Hudson et al. where a set of occluders is dynamically selected to compute occlusion in the scene [7]. Depending on this fact, viewer cannot see the occludee if the ocludee is in the shadow frustum of the occluder. Shadow frustums of the selected occluders are used to cull the scene. Culling is done for bounding boxes from top to down on the object hierarchy of the scene. If a node in the hierarchy is found to be totally occluded it is discarded for the current frame. If a node in the hierarchy is found to be partially occluded, testing is continued for the lower levels of hierarchy to find totally occluded or totally visible nodes.

Bittner et al. improved Hudson's approach by employing an occlusion tree built by combining shadow frustum of the occluders [8]. Frustums are combined in the way presented by Chin and Feiner [9]. Once the tree is built as explained in

Bittner's study, instead of testing the scene against each of the shadow frustums, it is tested against a tree. Figure 4 is illustrates the method.

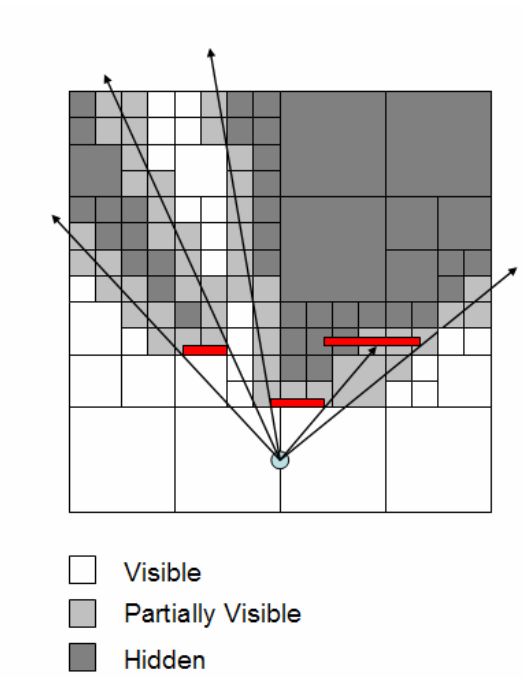

Figure 4 Illustration of Bittner's method.

A group of point-based methods perform culling on the image during rendering. Objects fill out the image during rendering and consequent objects culled away using the filled parts of the image. Algorithms in this group operate on a fixed array of pixels and therefore are more robust and easier to implement.

Greene enhanced standard Hidden Surface Removal method Z-Buffer to Hierarchical Z-Buffer (HZB) by adding multiple layers of different resolutions

[10]. Each level on the hierarchy is constructed by halving the resolution of lower level on each dimension. Z value associated with an element is the furthest z value of the elements forming that element on the lower level. During scan-conversion change of a z-value on leaf elements are propagated to upper levels.

Greene presented the scene as an octree [10]. Nodes of octree are traversed from top to down and from front to back and tested against depth values in hierarchical depth buffer. If a node is found to be occluded it is discarded, otherwise testing continues for its children. To test a node, its faces are tested against z buffer hierarchically starting from the coarsest level. If a primitive is found to be further than the value in z-buffer it is occluded, if not testing continues in the lower levels of z-buffer until the visibility can be decided. Objects in non-occluded leaf nodes are rendered and the z-buffer is updated. Greene proposed modifications on graphics hardware for hardware implementation of hierarchical z-buffer.

Zhang proposed a method which is similar to Greene's Hierarchical Z-Buffer [11]. To work on current graphics hardware, method separates occlusion overlap test and depth test. The method utilizes a data structure called Hierarchical Occlusion Map (HOM). HOM stores only opacity information while the depth information is kept separately. At rendering time, for each frame HOM is constructed and scene is culled with it. HOM is built by rendering a set of occluders on the frame buffer. For building the HOM only opacity information is needed. Therefore all features like lighting and texturing are turned off and occluders are rendered s pure white on black background. After rendering on frame-buffer, a hierarchical structure is built like the HZB. Elements on levels upper then the finest level can have gray opacity values to

reflect opacity of the region that is represented by the element. To test occlusion of an object, first its bounding box is projected on the HOM at the appropriate level and tested against opacity values for overlap with occluders. Then the depth values are used to decide whether the object is occluded or not.

A different point-based image-space approach is proposed by Wonka et al. for urban environments [12, 13]. Wonka exploited the fact that urban environment is in fact 2.5D. In other words the scene can be described as $z = f(x, y)$. Volume behind the occluder limited with the planes generated by the viewpoint and the edges of the occluder is called Occluder Shadows. Projection of occluder shadow on the floor is called the footprint and used to compute occlusion. An ocludee is hidden if it lies in the footprint of the occluder and it is below the topmost edge of the occluder assuming that viewer is located below the occluder heights. The z-buffer is used to create shadow footprints and their height information by rendered with an orthographic top view of the scene as in Figure 5. Occluder shadow footprint..

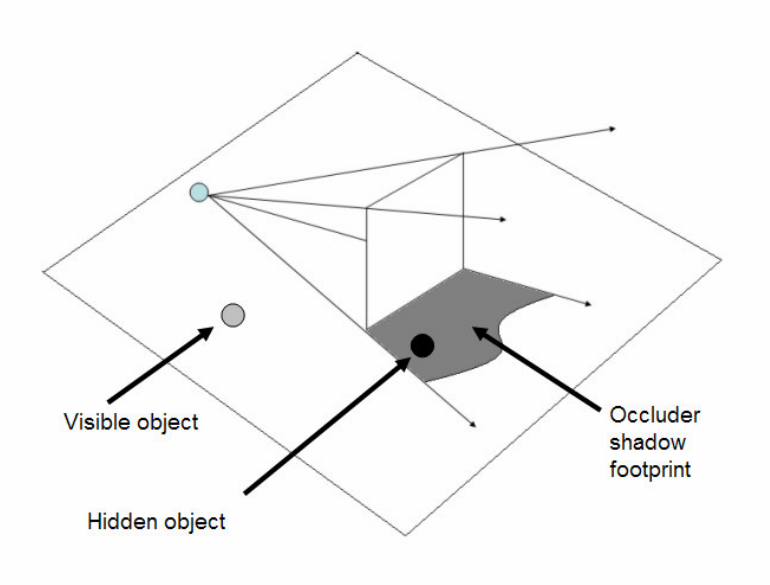

Figure 5. Occluder shadow footprint.

Also Downs et al. exploit the 2.5D characteristic of urban environment [14]. They use the abstraction called Occlusion Horizon in their algorithm. OH is the connected set of lines passing through the top of the visible buildings in urban scenery. Downs maintained OH as a binary tree for optimum performance. Figure 6. Occlusion Horizon in Urban Scenery and its binary tree representation. illustrates occlusion horizon in urban scenery and its representation as binary tree. Downs use approximations of inner and outer volumes of objects in the scene. For outer volume approximation bounding box is used. For inner volume approximation, a set of convex vertical prisms (CVP) that fit in the object volume is used as in Figure 7. Building and CVP.

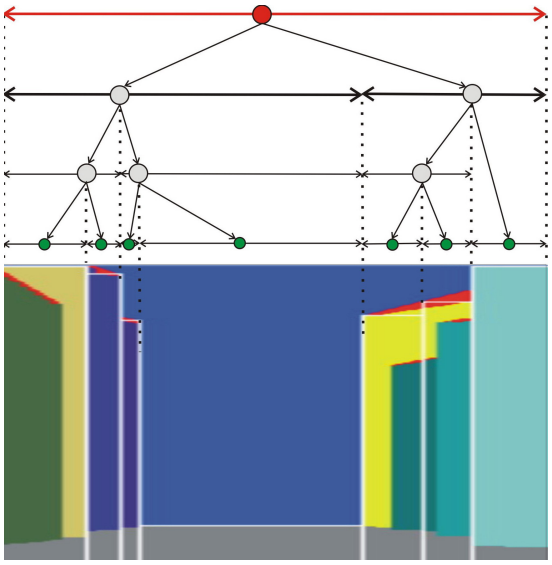

Figure 6. Occlusion Horizon in Urban Scenery and its binary tree representation. (Courtesy of Koldas)

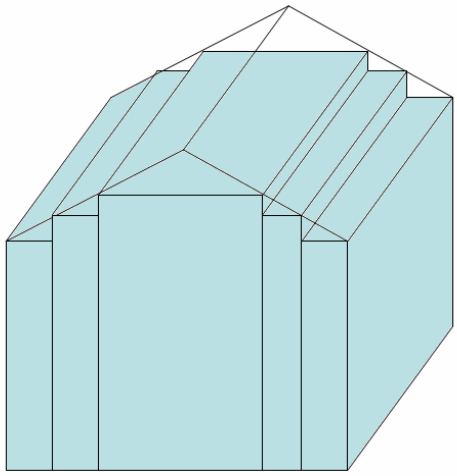

Figure 7. Building and CVP.

The method assumes a plane which sweeps the scene starting from the viewer and through the viewing direction. Scene is accessed hierarchically. Potentially visible objects are compared with the horizon when the sweeping plane touches them and their CVPs are used to update the horizon when the sweeping plane leaves them. By this was accidental occlusion of an object by another which does not occlude it at some point. Outer volume approximations of objects are used for comparison and inner volume of objects are used for updating the horizon. If outer volume of an object is found to be below the horizon it is occluded since the objects are traversed from front to back. Sweeping plane abstraction is implemented by an event queue.

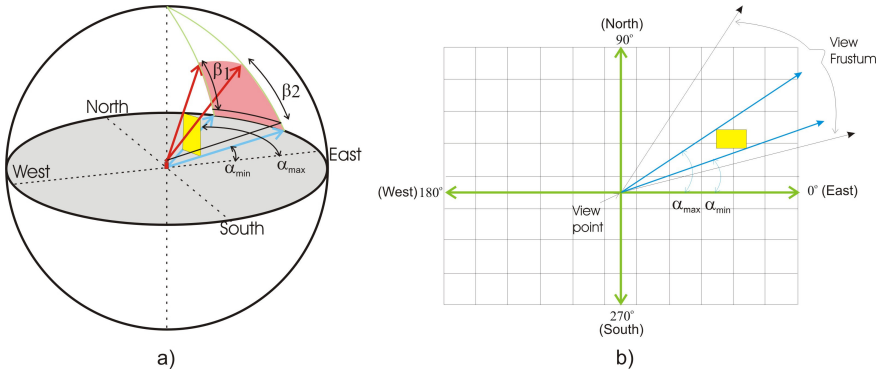

Figure 8. Occluder Shadow in Polar coordinates. (Courtesy of Koldas)

Koldas et al. improved the occlusion horizon method with incremental updates and polar coordinate system as called the method as Delta-Horizon [15]. Instead of recomputing the horizon for each position of the viewer, horizon is updated. Koldas exploit the fact that the occlusion horizon and PVS do not change much during a walkthrough and parts of the horizon to be updated can

be identified based on the horizon and the direction and size of the viewer movement. Instead of using image space, Koldas utilized polar coordinate representation of occluder shadows and the horizon as in Figure 8. Occluder Shadow in Polar coordinates.. Figure 9. Delta Horizon summarizes Koldas's method.

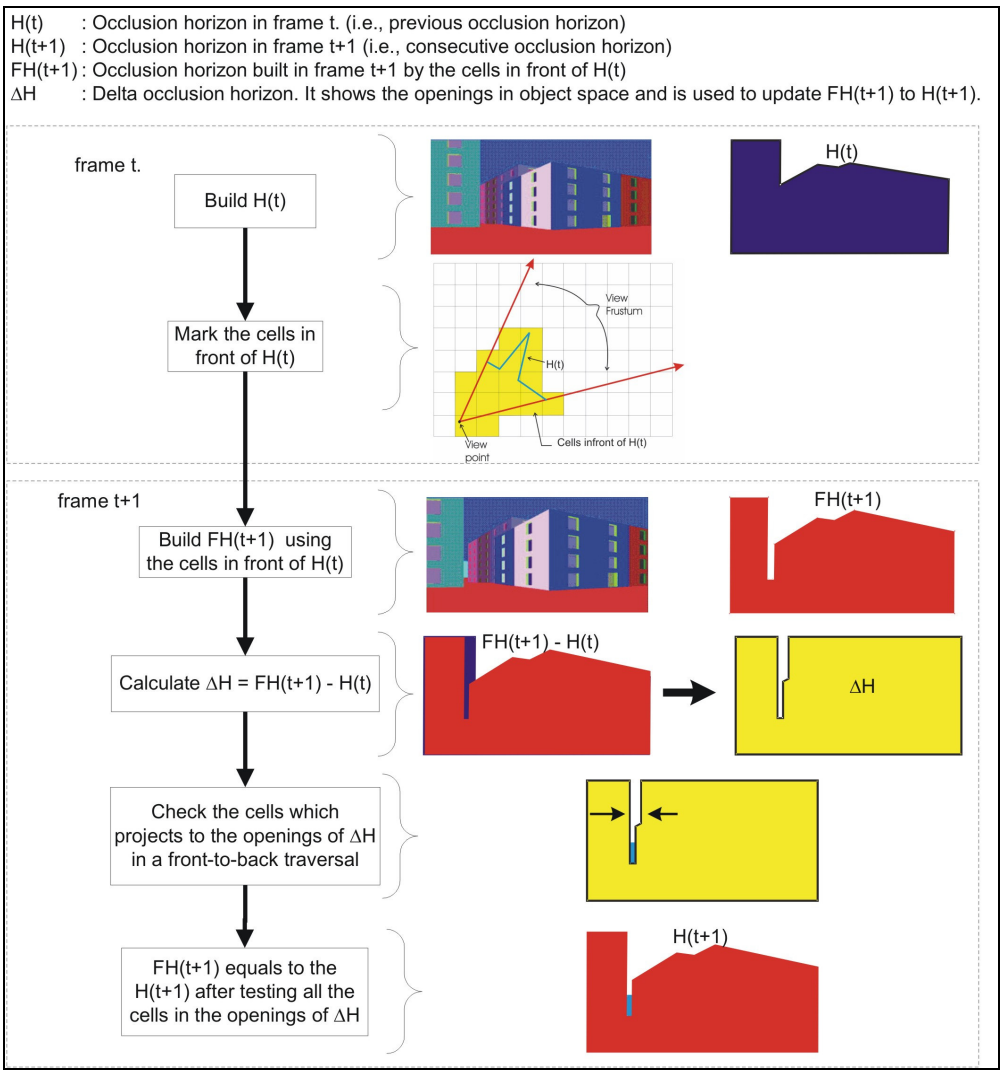

Figure 9. Delta Horizon method. (Courtesy of Koldas)

2.2 OFFLINE VISIBILITY CULLING ALGORITHMS

Offline visibility culling methods generally partition the scene into cells and compute the PVS of each cell at preprocessing. At run-time, only the objects in the PVS of the cell where the viewer is located are rendered. Methods computing the PVS for cells of the scene are classified as region-based or from-region visibility culling methods [4]. PVS of a cell is valid for the frames generated while the viewer in that cell. However it is hard to compute exact or conservative PVS for regions. One basic approach is sampling the visibility from viewpoints inside the cell and building an approximate PSV; but approximate PVS may cause unwanted flickering. Moreover, region based visibility is not effective with occluders smaller than the cell since an occluder smaller than the cell generates only a finite conservative shadow frustum behind it as seen in Figure 10. a) Occluders smaller than the cell. b) Aggregate umbra.. Methods based on large occluders were proposed by Cohen and Vazquez. However occluders are usually smaller than the cells in real applications [4, 16, 17].

The important concept that overcomes this problem is using the aggregation of the occlusion caused by the individual objects. This approach is called occluder fusion. Individuals forming the aggregate umbra do not have to be connected or convex [4]. A sample aggregate umbra generated with this approach is shown in Figure 10. a) Occluders smaller than the cell. b) Aggregate umbra. Effective methods which are significantly more successful than the previous methods were developed with utilization of occlusion fusion concept [4].

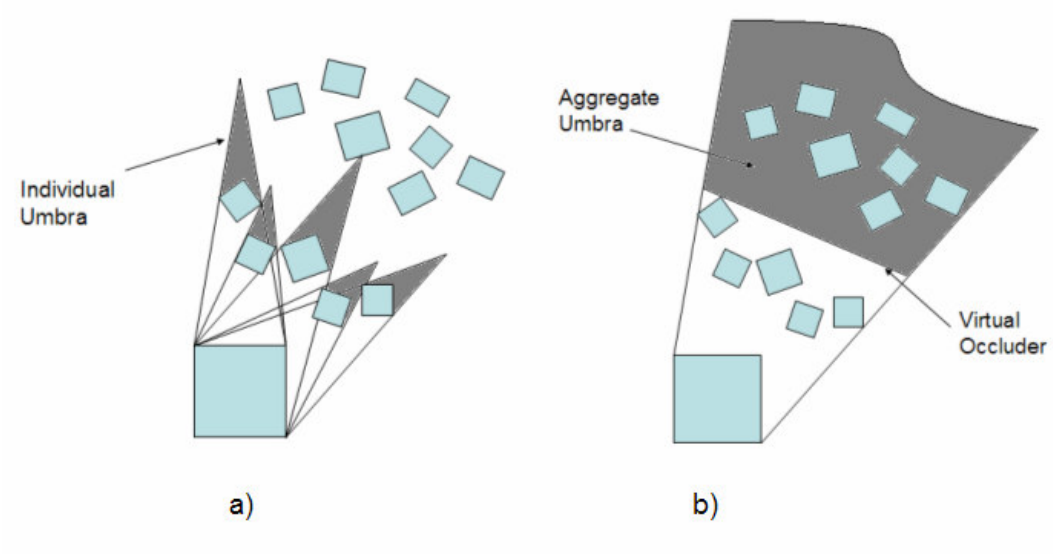

Figure 10. a) Occluders smaller than the cell. b) Aggregate umbra.

Teller builds a BSP to divide the scene into convex cells. Boundaries of the cells are composed of large opaque faces like walls. Non-opaque portals on the cell boundaries are identified and used to build an adjacency graph where the nodes are the cells and the portals are the edges. If a line from a point in a cell to a point in another cell, only passing through the portals thus not intersecting with cell boundaries, then these two cells can see each other through portals. These lines are called sightlines and with adjacency graph utilized to build portal sequences. PVS of a cell is the cells which are visible from the cell, are in the view-frustum, all cells in the portal sequence are in the frustum and there exists a sightline in the frustum. Teller extended his work to 3D and arbitrary portals. [18]

Schaufler's method [19] uses discrete representation of space. First the object boundaries are rasterized into discrete space and interior of these boundaries filled with opaque voxels. For each view-cell adjacent voxels are grouped into larger occluders. Regions of space hidden by these occluders are marked as occluded. Regions are extended to space and occluder fusion is applied. To determine visibility of an object, the space occupied by the object is tested against the occluded volume of the scene for the cell where the viewer is located.

Durand developed a conservative visibility preprocessing with use of extended projections which works on image-space [20]. Basically occluders and occludees are projected on a plane. If projection of an occludee is completely covered with aggregated projections of occluders then the occludee is said to be hidden. To ensure conservativeness projection of occluders are underestimated and projection of occludees are overestimated by extended projection. Extended projections are represented with a discrete extended depth map. Positions and number of planes are crucial for the effectiveness of the method and proper decision is also explained in the study.

Koltun proposes using virtual occluders which is an aggregated representation of occlusion generated by multiple occluders [21]. In the preprocessing stage, nearby objects are iteratively added to clusters of objects which then used to build the virtual occluder that satisfies the occlusion generated by the objects forming it. Virtual occluders are placed at the furthest point of the cluster generated it. At the end of the iterations there are many virtual occluders which can be represented by a set of much smaller and effective virtual occluders. At run-time Scene is hierarchically tested against the virtual occluders. Since the number of occluders is small the method runs fast.

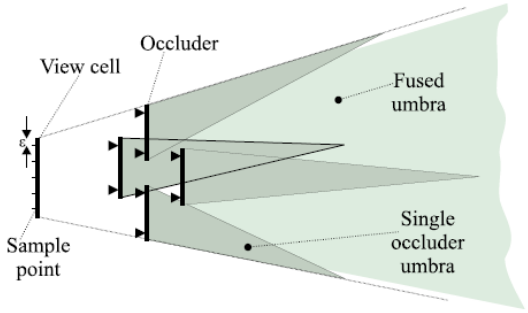

Figure 11. Occluder shrinking. (Courtesy of Koldas)

Wonka present a method to compute conservative approximation of visibility from a cell with discrete sampling from that cell [95]. He observed that if an occluder is shrank by ε, occludees occluded by the shrunk occluder, stay occluded with respect to the original sized occluder while moving the viewpoint no more than ε. Depending on this observation, visibility from a cell can be approximated by shrinking the occluders by ε and taking samples on the boundary of the cell with intervals of ε. Sampling is visualized on Figure 11. Occluder shrinking.. Size of ε generates a trade-off between cost of visibility calculation and accuracy of the computed PVS. Small ε increases accuracy and computation cost while large ε does vice versa.

2.3 PREFETCHING

Region based approaches are necessary for server-client applications and application with huge data sets. However they have their own drawbacks. Preprocessing time and storage requirements are high. Moreover a transition from one view-cell to another causes loading bulk data necessary to build PVS of the destination view-cell. During loading either the rendering pauses or the scene is rendered with incomplete data. Stall can be overcome by prefetching PVS of adjacent cells before leaving the current cell provides smooth visibility. The prefetching methods that this study concentrates on are visibility based prefetching methods.

Both Teller [23] and Zach [24] fetch scene data in small chunks of different LOD representations. Decision of data to be retrieved and level of detail is made according to some benefit/cost calculation. Parameters like viewing direction and view position used for fetching data are estimated for future based on short history of viewer motion to predict the future. Based on this prediction prefetching is performed.

Koltun utilizes a simple prefetching method for his dual-space visibility algorithm [25]. In this algorithm, with the PVS of the current cell, PVSs of all adjacent cells are also prefetched. When the viewer passes to an adjacent cell, PVS of that cell is prefetched already and rendering continues smoothly. Initially PVS of the cell containing the viewer and the PVSs of neighbor cells are fetched. Then when the viewer passes to a new cell all of the PVSs of new cell's neighbors are fetched.

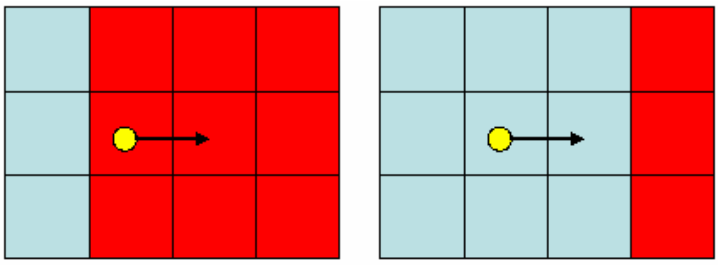

Figure 12. a) Prefetching with Koltun's approach b) Delta Transmission. Red cells are fetched during transmission.

Zheng and Chan improve Koltun's algorithm [26]. First enhancement they constructed is the delta-transmission. Instead of fetching all neighbors of a cell, they just fetch the PVSs of the cells which are not already in the memory. Figure 12. a) Prefetching with Koltun's approach b) Delta Transmission. explains delta-transmission. Instead of re-fetching six of the cells only three cells are fetched.

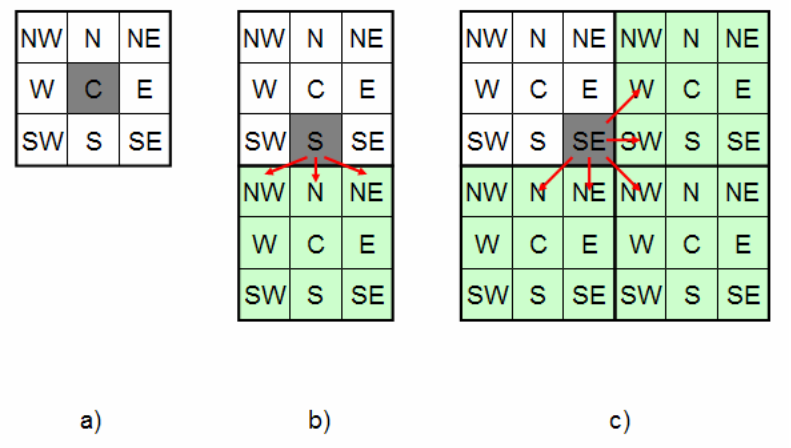

Figure 13. Monitoring viewer tendency for prefetching. a) No cells are prefetched when the viewer is in the middle (M) of the cell. b) South neighbor is prefetched when the viewer is in the south (S) of the current cell. c) South, South-East and East neighbor cells are prefetched when the viewer in the north-east (NE) of the current cell.

Key point of their algorithm is sub-dividing the cells into subcells and avoiding redundant prefetching. They defer prefetching until the viewer shows tendency to change view-cell. Also they avoid prefetching all eight neighbors by deciding a subset of neighbor cells according the tendency of the viewer. Figure 13. Monitoring viewer tendency for prefetching. a) No cells are prefetched when the viewer is in the middle (M) of the cell. b) South neighbor is prefetched when the viewer is in the south (S) of the current cell. c) South, South-East and East neighbor cells are prefetched when the viewer in the north-east (NE) of the current cell. summarizes their approach.

CHAPTER 3

THE PROPOSED PREFETCHING METHOD

To render complex urban data, scene data is culled to decrease amount of data sent to rendering pipeline. Culling is done in point based or region based manner. While point based methods calculate Potential Visible Set for each location of viewer, region based methods partition the scene in view-cells and calculate PVS of each view-cell at preprocessing. Therefore, region based approach removes overhead of PVS calculation from rendering time. However a transition from one view-cell to another causes loading bulk data necessary to build PVS of the destination view-cell. During loading either the rendering pauses or the scene is rendered with incomplete data.

This section proposes a method to avoid stalls generated by the loading of the scene data during view-cell transitions. General approach of avoiding stalls is called Prefetching. Prefetching is predicting next transition and loading necessary data before the transition occurs. Our method is based on the graph representation of view-cells and the transitions among them. A generic optimization algorithm, which is called Simulated Annealing, is utilized to optimize prefetching of scene data on our graph representation.

The proposed method for pre-fetching scene data improves smoothness of view cell transition in region based rendering of complex urban environment. The method represents view-cells and the transitions among them as a graph and constructs abstract groups of view-cells. Groups are used for prefetching scene data. To construct groups, a generic optimization algorithm is used. The method

is customized by changing the heuristics and evaluation criteria used for grouping and constructing prefetching set. With the flexibility and abstraction provided by the method, better policies can be developed.

3.1 STATEMENT OF PROBLEM

Considering a transition T from view-cell S to view-cell D, four sets are defined where b_i is a building:

$PVS(S) = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}\}$
$PVS(D) = \{b_1, b_4, b_6, b_9, b_{11}, b_{12}, b_{13}, b_{14}, b_{15}, b_{16}\}$.
$discard(T) = \{b_2, b_3, b_5, b_7, b_8, b_{10}\}$
$load(T) = \{b_{11}, b_{12}, b_{13}, b_{14}, b_{15}, b_{16}\}$

Assuming *PVS(S)* is already in memory at the time of transition; buildings which are in *PVS(S)* and not in *PVS(D)* should be discarded. The set of buildings to be discarded is *discard(T).* Buildings which are in *PVS(D)* and are not in *PVS(S)* should be loaded. The set of buildings to be loaded is *load(T).*

It is clear that a break in the smoothness (a stall), whose length varies with the complexity of the building models, will occur. To overcome stalling, *load(T)* should be loaded into the memory before the transition takes place.

Statement of the problem proposes a structure. When view-cells are taken as nodes and transitions are taken as edges between nodes, a graph structure is

formed. Figure 14. View-cells to graph mapping. shows the mapping from view-cells and transitions to graph.

On this graph groups of nodes are defined according to some set of heuristics like "*nodes with transitions of probability higher than 50% should be in the same group*". Each these groups are assigned a set of buildings which is again built according to a heuristic like "*10 most expensive buildings should be included in the prefetching set*". When a transition occurs, first the PVS of the target view-cell should be constructed by fetching the missing buildings. While the viewer stays in the cell, perform prefetching of buildings identified in the set assigned to groups which includes the current view-cell/node. This simple schema constructs the basics of our approach.

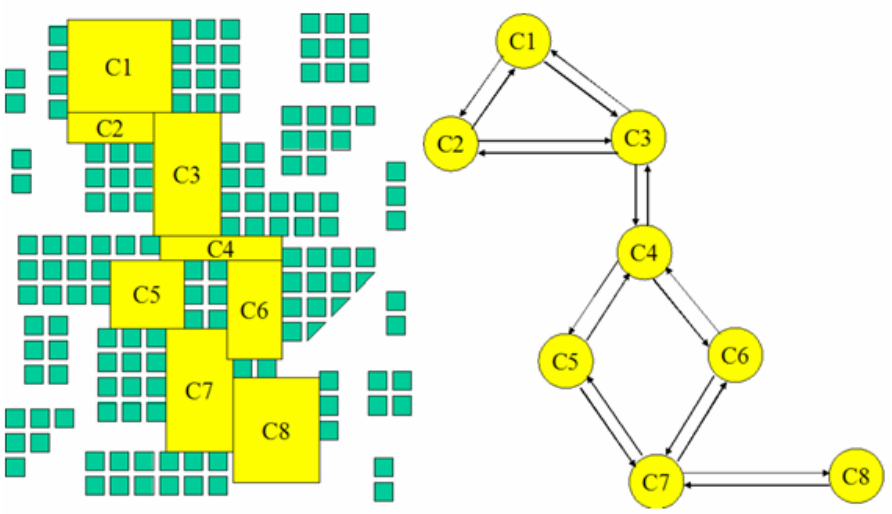

Figure 14. View-cells to graph mapping.

With this approach, optimizing prefetching is converted to graph partitioning problem. To generate useful partitioning which minimizes the transition costs we utilized a generic optimization algorithm called Simulated Annealing. There are three main steps of this approach. The first step is to representing the scene as a graph. The second step is to map the prefetching process to a graph partitioning. The third step is to solve graph partitioning problem using simulated annealing algorithm by utilizing heuristics. At the end, whole prefetching optimization problem is transformed to defining effective heuristics of graph partitioning.

3.2 SIMULATED ANNEALING

Simulated annealing (SA) is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space.

Simulated annealing builds an analogy between physical systems and optimization problems. Point *s* in the search space of the problem is analogous to one state of the physical system. Internal energy of the physical system at current state is analogous to the cost of a solution, *E(s)*, in the search space. The goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy. Speaking in terms of optimization problem, this means finding the point in the search space with minimum *E(s)*.

Basically SA works in five steps which are repeated as iterations. Iterations are repeated until the computation budget is exhausted. In SA, computation budget

is realized with a global variable called Temperature. Temperature is gradually decreased in each iteration. Five steps of simulated annealing are:

1. Choose a neighbor state *s'* of the current state *s*.
2. Calculate *E(s')* the energy of the candidate state.
3. Store *s'* if it is the best solution so far.
4. Decide moving to *s'* or staying at *s*.
5. Cool down – decrease Temperature.

It is shown that, for any given finite problem, the probability that the simulated annealing algorithm terminates with the global optimal solution approaches to 1 as the annealing schedule is extended.

Which separates SA from greedy algorithms is the function used for step 4. Decision is made by a probability function of acceptance P(e, e', T) where e = E(s), e` = E(s`) and T = Temperature. P is implemented so that it allows moving to worse states depending on T and the difference between energies of the candidate state and the current state. Probability of moving to a state with higher energy (worse state since we are trying to minimize the energy) increases with increasing T and decreasing difference in energies of two states. On the other hand P is always 1 when candidate state has a lower energy. This property of SA avoids being stuck at local minima by allowing it to jump out of the local minima to a state (probably with higher energy when T is high) which can lead to global minima.

Lack of the definition of exact solution leads to utilization of stochastic optimization methods. Alternatives in this area include Genetic Algorithms, Ant colony optimization and those similar to simulated Annealing like Stochastic

Tunneling and Tabu Search which traverse neighbor solutions of current one and try to avoid local minima. Although it is probable to achieve optimization with alternatives, simplicity of understanding and implementation, resulted with utilization of Simulated Annealing.

3.3 METHOD

3.3.1 REPRESENTATION

Our method is built on top of a representation of scene as a graph. View-cell graph is defined as <N, E> where N is set of view-cells and E is set of transitions $\{t_1, t_2, t_3\ldots\}$. For N each cell has set of buildings $\{b_1, b_2, b_3...\}$ which has to be rendered when viewer is in that vie-cell, in other words the PVS of the cell. A transition is defined as <S, D, L> where S is source view-cell, D is destination view-cell and $L = \{b_1, b_2, b_3\ldots\}$ is a set of buildings to be loaded.

When a transition <S, D, L> occurs, it is sufficient to load buildings in L to built necessary building set, PVS of the destination cell, to render. L is the set of buildings which are not in the PVS of S and are in the PVS of D. Therefore each edge/transition has a cost which is the loading time of the buildings that will be loaded for completing PVS of destination view-cell.

3.3.2 MAPPING

Costs of the transitions cause the stalls and that should be minimized. Assuming there is more resource than necessary for rendering PVS of a regular cell, some more IO can be performed in parallel while the viewer stays in a cell. Deciding which data to be loaded drives us to implement a prefetching policy.

Simplest application of prefetching would be loading PVSs of all neighbor cells. This approach is static in terms of customization and prioritization. It ignores scene specific properties like time spent in cells and size of the cells. When the neighbor cells are small or the viewer passes through a cell in a short time, next transition will again cause a stall. Moreover when there are more transitions than the memory limitations can handle, some of the transitions should have a priority for prefetching to increase throughput of the prefetching. Solution of the problem should consider scene specific information, viewer behaviors and memory limitations.

Instead of defining strict rules of prefetching we developed a more flexible method based on representation of the scene as a graph. We define groups of adjacent nodes and construct sets of buildings for each of these groups and call these sets as PS (Prefetching Sets). Idle time spent in a cell is used for loading PSs of the cell.

Figure 15. Sample partitioning illustrates a sample partitioning of a graph. It is seen that groups are allowed to intersect with each other. When the viewer is in a node which is included in more than one group, the PS to be used for Prefetching is constructed based on the union of building sets of these groups.

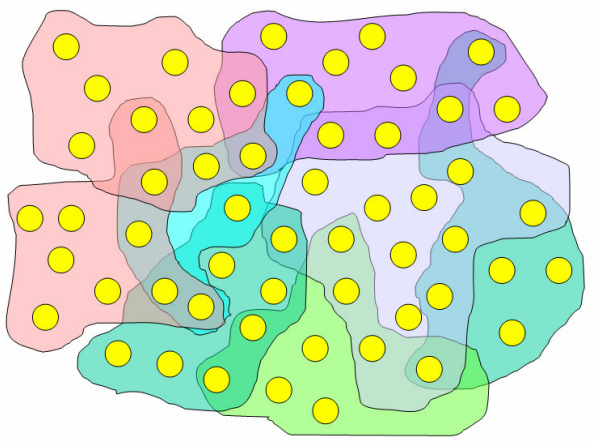

Figure 15. Sample partitioning

Definition of optimum changes with the objective. For example minimizing the average transition cost may be an objective. Heuristics are used to optimize the grouping of cells according to the defined objective. Examples of these heuristics are:

- High probability transitions pull source and destination nodes into the same group.
- Cells where the viewer stays for short time act like transitions between their neighbors.
- Central cells with number of transitions above a defined threshold pull its neighbors.

First heuristic is intuitive since it is clear that if there is a high probability for the viewer to pass to a neighbor cell than content of that cell should be prefetched. Second heuristic exploits the fact that when the viewer stays short

in a cell, policy should consider neighbors of this short stayed cell. Third one uses the natural grouping exposed by the structure of the scene. If there is a cell which is connected to lots of cells, it is clear that constructing a group around this cell will be advantageous.

Once a group is formed, next point to consider is selecting a set of buildings as PS of the group. Which buildings from the cells should be included in the PS of the created group? Answers may include the following items:

- All units of all cells.
- High cost buildings.
- Long and frequently appearing buildings.

First one is the simplest, keeping the PS as the union of PVSs. A PS of a group contains all the building contained by PVS of the participant cells. Second one is a smarter choice, prefetching the high cost buildings but the effectiveness of the choice is not clear since it is not guaranteed that these building are seen most of the time. Third one is building the PS as a subset of all buildings which only includes most active buildings, in other words those buildings which are rendered for the longest time.

Two points where heuristics are utilized were considered: forming groups of cells and building PSs for groups. Both steps are subject to inevitable constraints: memory size and loading time. Loading time constraint is explained first. Prefetching at a cell can continue as long as the viewer stays in that cell. When the user leaves the cell, prefetching is stopped for left cell and starts for the arrived cell. In a similar way, data exceeding the memory limitation can not be loaded. Total size of the building to be prefetched for a cell therefore cannot

exceed the memory size. Both of these constraints can be embedded in the implementation or just be ignored because of the algorithm utilized for optimization. Practically, ignoring the two constraint does not cause any problems for execution of the algorithm and the rendering but it is for sure that it affects the effectiveness of prefetching. Situation will be clarified when application of simulated annealing is explained later.

As stated but not clarified before, prefetching by grouping can be applied for one of the possible global objectives. Utilization for a generic optimization algorithm provides flexibility to work for different goals. An evaluation function is utilized as detailed later to calculate the score of a solution. Different scoring functions are proposed for three objectives:

- Minimizing maximum transition cost.
- Minimizing average transition cost.
- Minimizing popular path cost.

The first objective is used to reduce the cost, which is the stall time during a transition. If this value is kept below a specific value then the walkthrough runs smoothly. The second option optimizes for the throughput of the walkthrough. Minimizing the average cost of transition means minimizing the overall IO cost of the walkthrough. The third option is suitable for walkthroughs on low end resources. When the resources are limited, instead of the first two options, this option is utilized for optimizing costs for popular path, those are used mostly by the viewer. When a multi-viewer environment is considered depending on the facts of the scene and its semantics the third option provides focusing on the most effective transitions. An example where the third objective is applicable is that, a client-server application of hundreds of clients are connected to a server

and play an RPG game. At a specific location of the world, there is a monster that clients avoid facing it and run away instead of fighting. Even this one semantic property of the scene will create a popular path where the users come to the cell containing the monster and run away to the next cell containing the escape. It is wise to prefetch the escape path before the client faces the monster since probably he will take the escape path.

It can be argued that discarding unnecessary data should also be considered in this study. It is clear that groups generated by this method can be used for setting priorities for discarding data. Discarding with priority is stated as future work and out of scope for this study.

3.3.3 ADAPTATION

Previous section established a baseline for creating effective prefetching policy. Scene was represented as graph, prefetching was mapped to graph partitioning, evaluation methods were stated, heuristics used for grouping and PS construction were developed. If a mapping of our partitioning problem is generated, SA can be utilized.

Repeating steps of SA were stated in the previous sections as followings:

- Choose a neighbor state s` of the current state s.
- Calculate E(s`) the energy of the candidate state.
- Store s` if it is the best solution so far.
- Decide moving to s` or staying at s.
- Cool down – decrease Temperature.

Mapping starts with defining the search space for SA. For our case each partitioning of the view-cell graph is a solution. Therefore our search space/solution space is the set of all possible partitioning of view-cell graph.

In the solution space, the relation between neighbor solutions should be defined in order to have a basic step which leads to a new solution depending on another. A solution can be reached from another by doing one of the followings:

- Adding a view-cell to a group.
- Removing a view-cell from a group.

It is clear that starting from an arbitrary partitioning of the graph, with using only the two actions stated above, all search space can be traced.
Evaluation of a solution is the step after selecting a neighbor solution. Since SA tries to minimize the energy, it can be defined as E(s) = C(s) where C(s) is the cost to be minimized. Three options suggested as goals can be directly utilized as evaluation function. All of the three suggestions are straightforward to implement. For maximum cost minimization, C(s) is the cost of most expensive transition. For average cost minimization, C(s) = sum of transition costs / transition count. For popular transition cost minimization C(s) = sum of the costs of transitions included on the popular paths.

Comparing the new solution with the best solution so far and storing it, is trivial. In the original formulation of the method by Kirkpatrick *et.al* [27], the acceptance probability $P(e,e',T)$ was defined as 1 if $e' < e$, and $\exp((e - e') / T)$ otherwise. Although there is no mathematical justification for using this

particular formula in SA, this formula is quite popular. Therefore, fourth step of the algorithm is realized with this formula. At each step Temperature is decreased by one. With this simple approach T is the number of iterations.

Memory and time constraints were mentioned in the pervious section but the explanation was deferred to the end of this section. With the schema explained these constraints are embedded in the implementation at step one of SA, choosing a neighbor solution. While moving to a neighbor solution, if adding a node to a group violates one of these constraints, simply the candidate solution is discarded and another candidate solution is built. When constraints are not considered during candidate generation the drawback will be producing fake best solutions which cannot be realized during rendering. Since these constraints cannot be violated at rendering time, fake best solutions are still optimizations with less effectiveness. The sample implementation considers only memory constraint for simplicity.

CHAPTER 4

IMPLEMENTATION OF THE METHOD

Sample implementation works with a subset of suggested heuristics. Heuristic used for forming groups is:

- "*High probability transitions pull source and destination nodes into the same group.*"

Heuristic selected for constructing PS is:

- "*Long and frequently appearing buildings.*"

Implementation is tested for the goal:

- "*Minimizing average transition cost.*"

There are two major parts of the implementation:

1. Generic Simulated Annealing Implementation
2. Adaptation of Simulated Annealing for Prefetching optimization

Below is the pseudo code describing the implementation of simulated annealing:

```
CoolingSchedule         coolingSchedule;
SolutionSpace           solutionSpace;
StateTransition         stateTransition;
ScorePolicy             scorePolicy;
Solution                bestSolution;
Solution                candidate;

while (coolingSchedule.getTemperature() > 0) {
        candidate = solutionSpace.getCandidate();

        if (candidate better than bestSolution) {
                bestSolution = candidate;
        }

        candidateScore = scoringPolicy.score(candidate);
        currentScore = scoringPolicy.score(solutionSpace.getCurrent());
        temperature = coolingSchedule.getTemperature();

        if (stateTransition.checkTransition(temperature, currentScore,
        candidateScore)) {
                solutionSpace.acceptCandidate();
        }

        coolingSchedule.cool();
}
```

Cooling schedule keeps temperature and gradually decreases it with each call to cool function. Solution space keeps best solution available with current solution and generates candidate solutions. ***StateTransition*** probabilistically decides to accept the candidate solution as the current solution or not. Score policy calculates the score/energy of each solution. Implemented algorithms for each of these briefly mentioned components are explained in the next section.

4.1 ADAPTATION OF SIMULATED ANNEALING

Following figure show the overview of simulated annealing and its adaptation:

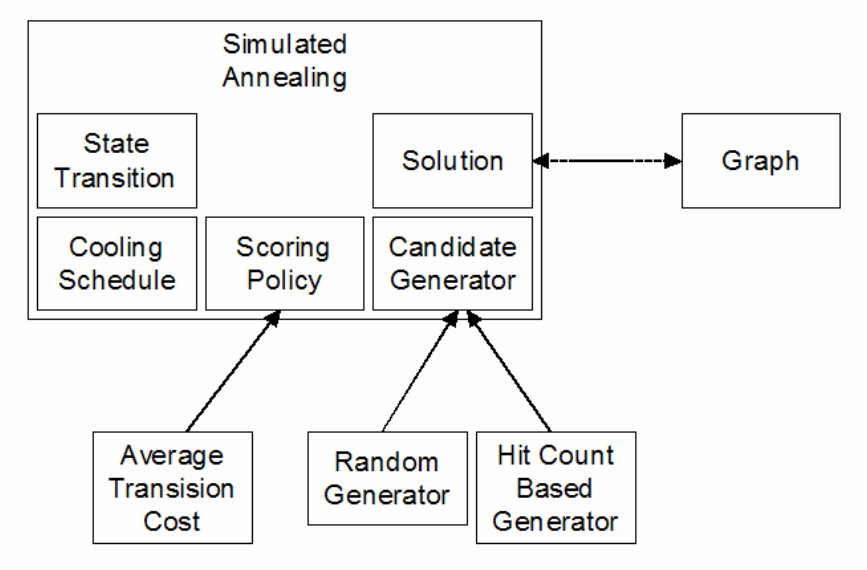

Figure 16. Adaptation Overview

Cooling schedule used for tests is a straightforward implementation. Cooling schedule is started with an initial temperature and in each iteration; the temperature is decreased by one. Literally, cooling schedule counts the number of iterations and makes simulated annealing to stop after specified number of iterations.

State transition function $P(e,e',T)$ is defined as 1 if $e' < e$, else $exp((e - e') / T)$ where e and e` are the scores assigned with the associated Scoring instance and T is the current temperature. This allows moving to worse states depending on T and the difference between energies of the candidate state and the current

state. Probability of moving to a state with higher energy (worse state since we are trying to minimize the energy) increases with increasing T and decreasing difference in energies of two states. On the other hand P is always 1 when candidate state has a lower energy. This property of SA avoids being stuck at a local minimum by allowing it to jump out of a local minimum to a state (probably with higher energy when T is high) which can lead to global minima.

A solution is defined as a set of groups of nodes in the graph and sets of buildings (Prefetching Sets) associated with groups. Partitioning is not used in as the original definition but used as in the relaxed definition which allows intersecting partitions.

Solution space keeps the best solution available, the current solution and generates candidate solutions in two alternative ways. First one is called the default candidate generator and randomly selects nodes and groups to generate candidates.

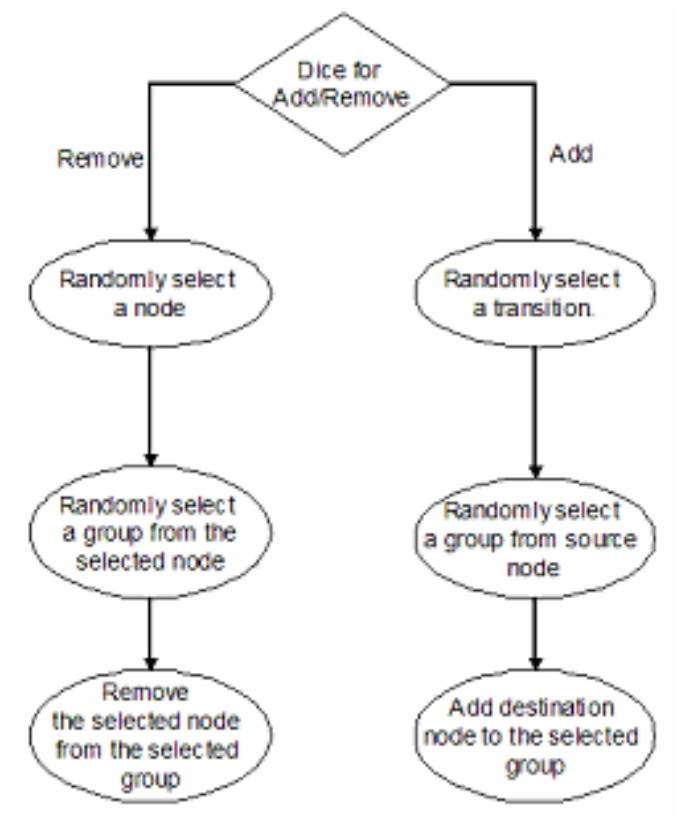

Figure 17. Random candidate generator

Second way is supported with heuristics to create tendency of generating better solutions for optimizing the average transition cost. While adding a node to a group, transitions with higher hit count are favored. While removing a node from a group, transitions with lower hit count are favored. After a transition is selected, its source and destination nodes are used to generate a candidate. Since method is developed for prefetching, destination nodes are added to and removed from group of source node. Since the implementation allows multiple groups on one node, group is selected randomly among the available groups.

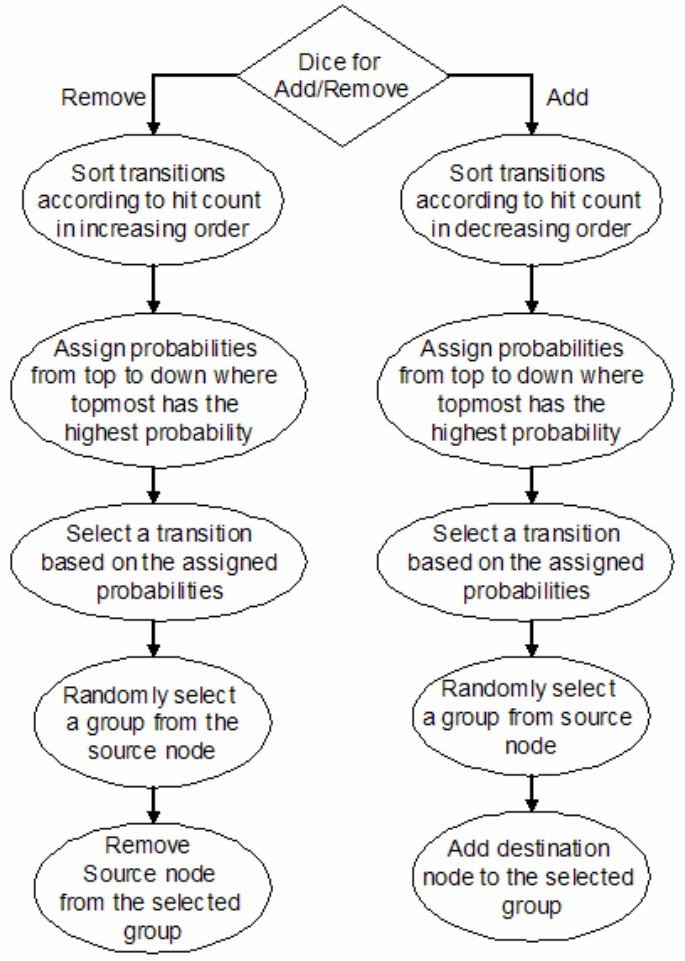

Figure 18. Candidate generation with heuristic

After a group is modified its PS should be updated. Two alternative methods for PS construction are implemented. The first method randomly chooses buildings among all buildings of nodes forming the group. The second method favors buildings from nodes with higher hit counts. However, the heuristic used

for the e second method does not perform better. See CHAPTER 5 for details and discussion of the situation.

Sample implementation tries to minimize the average cost of transitions; therefore the scoring policy is implemented to compute the average cost of transitions. The straightforward formula is Σ *cost(t) / number of t* where *t* is a transition.

4.2 CITY DRAWER

Figure 19. Screenshot of CityDrawer

To generate test data a utility called City Modeler is also developed. City modeler displays images as background and allows defining buildings and view-cells as polygons. City Drawer can save and load city data as text files. Format of the text files are given below.

4.3 GRAPH GENERATOR

Another utility used after the generation of city data by City Drawer. Graph Generator takes output of the City Drawer and constructs a graph with it. Then Graph generator saves the graph data to another text file. At the end, there are three files: View-Cells, Buildings and Graph. These form the input for the algorithm.

4.4 TEST BED

Algorithm runs with statistical data of transitions between cells. The assumed real environment of the method is an application with hundreds of users, where the prefetching policy is continuously optimized with the statistical data generated by the users. Method tracks user trends in the environment and updates the prefetching according to these trends. However, generating the input and testing the results requires a simulator since it is not possible to gather hundreds of people to be used as testers. A simulator of tourists touring the city is utilized for the purpose.

There are tourists and bad guys in a city. Tourists are far more crowded than the bad guys like 1/100 or 1/1000. Tourists want to see as many buildings as possible, stay in the same view-cell with the bad guys as short as possible and they chat with each other. When tourists face each other in a cell, the time they spend in the cell increases. When there is a bad guy in the cell the tourists leaves the cell in a very short time. When the view cell contains only one tourist then the tourist stays in the cell proportional to the size of the cell since they discover every square meter of the view-cell. When the time is up and the tourist leaves a cell, it decides the destination cell with a probability generated according to cell PVS sizes. The more buildings in a neighbor cell the more it is likely that the tourist goes to that cell.

Simulator runs in iterations and at each iteration every tourist takes a single move to cell other than the cell it is in. When leaving a cell, they register their stay time to cell according to the scenario described above. When a tourist makes a transition it also increases the hit number of the cell transition it takes.

With the described simulator, transitions have probabilities assigned to them and cells have average stay times. These statistics are used for testing of the sample implementation.

CHAPTER 5

RESULTS AND DISCUSSION

Tests are performed on the previously described test bed with 100 tourists, 93 nodes and 135 buildings. Input statistics are collected with a run of 1000 turns where each tourist takes a move. On the total there are 100x1000 = 100000 view-cell transitions. Each row on the following tables represents a single test run. Each test run consists of three steps:

1. Create input statistics.
2. Run the method.
3. Collect output statistics.

Each building has an associated cost with is proportional to the area covered by it. This cost is utilized as the memory occupied by buildings when fetched. An upper limit defined for the fetching which is identified as cache size on the results tables.

Columns on the tables have the following meanings:

CACHE SIZE : Maximum allowed cost of buildings to be kept in building cache.

INITIAL TEMPERATURE : Number of cycles that the method will be run.

AVERAGE COUNT B.A. : Average number of buildings fetched in a view-cell transition before applying the method.

AVERAGE COST B.A : Average total cost of buildings fetched in a view-cell transition before applying the method.

FIRST SCORE : Initial score/energy calculated by the method.

LAST SCORE : Best score calculated by the method, i.e. the score/energy of the best solution.

AVERAGE COUNT A.A. : Average number of buildings fetched in a view-cell transition after applying the method.

AVERAGE COST A.A : Average total cost of buildings fetched in a view-cell transition after applying the method.

COST REDUCTION : Decrease in the average cost of a transition with application of the method in terms of percentage.

There is no specific unit of Cache Size and Building Costs. Since the tests are performed to see reduction ratios, only the magnitude of these value are important. It can be argued that the test data is limited in size. Since the local visibility is independent of the size of the complete data, the results reflect a good approximation to higher data sizes.

Although the method is assumed to be an off-line method, presenting its computational cost might be helpful. 800 iterations with full data set which is

mentioned above takes below 2 seconds. Even though there is no formal complexity analysis and comprehensive performance test available yet, parameters affecting the computational cost can be stated.

1. Number of buildings in the scene.
2. Number of nodes in the scene.
3. Number of iterations of SA.
4. Number of transitions in the scene.
5. Complexity of the utilized heuristics.

Each of the first four factors is expected to affect the computation time linearly or near linearly depending on the fact that the SA algorithm jumps randomly among nodes and transitions instead of tracing them one by one. However, suggested heuristics rely on sorting of transitions and nodes according to various criteria. Therefore, the performance of the method becomes dependant on the first four factors indirectly through the heuristics.

5.1 RESULTS WITH HEURISTIC CANDIDATE GENERATOR AND RANDOM PS CONSTRUCTOR

The first set of results are collected with CandidateGeneratorForAverage as candidate generator, ScoringForAverage as scoring policy and AllBuildingsPS-Constructor as prefetching set constructor. See CHAPTER 4 for details of these components. This set illustrates the effect of using a simple heuristic of effectiveness of the method.

Table 1. Results with Heuristic Candidate Generator and Random PS Constructor

CACHE SIZE	INITIAL TEMPERATURE	AVERAGE COUNT B.A.	AVERAGE COST B.A.	FIRST SCORE	LAST SCORE	AVERAGE COUNT A.A.	AVERAGE COST A.A.	COST REDUCTION
10000	25	12.12	5936	5066	4968	11.72	5829	1.80
10000	50	12.02	5920	5066	4815	11.86	5916	0.06
10000	100	12.04	5898	5066	4367	11.97	5925	-0.45
10000	200	12.02	5934	5066	3511	11.76	5872	1.04
10000	400	12.08	5956	5066	3160	11.82	5904	0.87
10000	800	11.98	5905	5066	2681	11.82	5890	0.25
50000	25	11.88	5878	5066	4942	11.51	5759	2.02
50000	50	11.04	5899	5066	4703	10.56	5379	8.81
50000	100	12.06	5942	5066	4493	8.64	4443	25.22
50000	200	12.04	5906	5066	3673	3.83	1998	66.17
50000	400	11.86	5895	5066	2888	1.99	998	83.07
50000	800	11.96	5908	5066	2661	1.83	785	86.71
100000	25	11.95	5875	5066	4905	11.77	5863	0.20
100000	50	12.03	5929	5066	4820	10.95	5474	7.67
100000	100	12.00	5883	5066	4441	8.79	4452	24.32
100000	200	11.88	5848	5066	3303	2.75	1446	75.27
100000	400	11.93	5897	5066	2991	1.30	762	87.07
100000	800	11.93	5900	5066	2715	0.91	490	91.69

Simulated Annealing implementation increases its effectiveness with increasing temperature as expected. However speed of decrease also decreases since the

probability of finding better solutions decreases. In other words score converges to some lower bound. (Figure 20. Performance of SA)

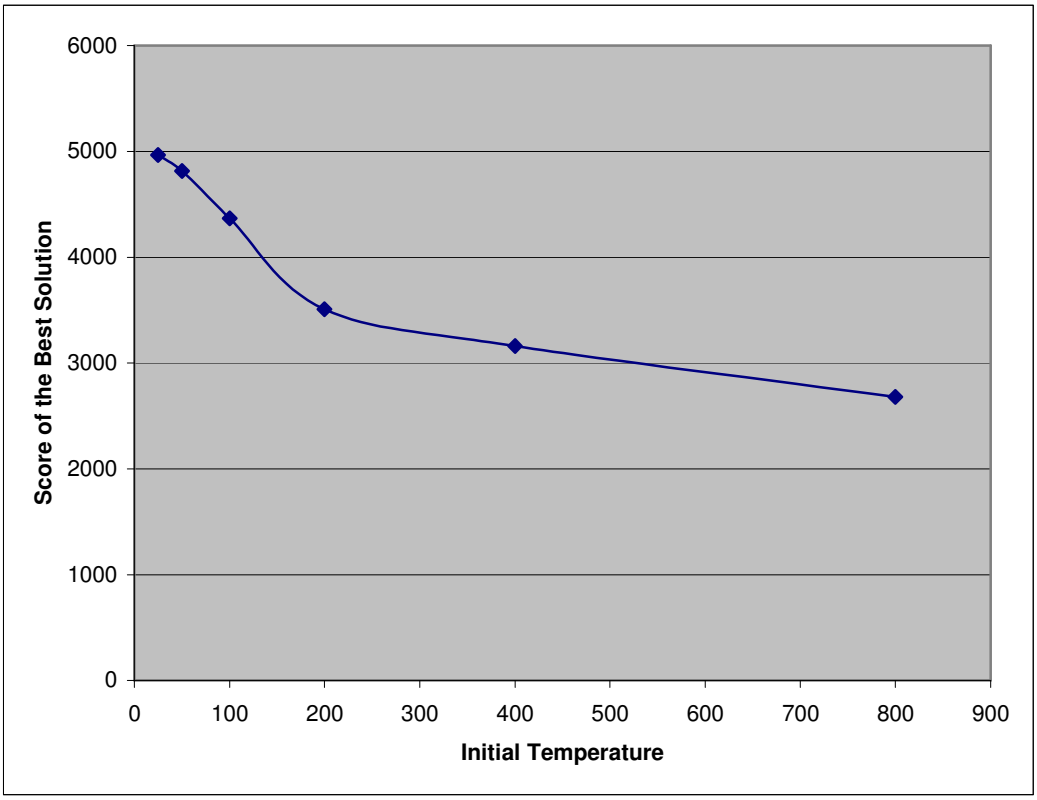

Figure 20. Performance of SA with Heuristic Candidate Generator and Random PS Constructor

Cost reduction column shows much more dramatic facts. Even though SA can only reduce the score to its half, effect of this reduction to view-cell transition cost is more effective. Cost reduction is strongly bounded to cache size. As seen from the results, while reduction is near zero with low cache sizes it can hit to

96% with high cache sizes. (Figure 21. Cost Reduction with changing Cache Size

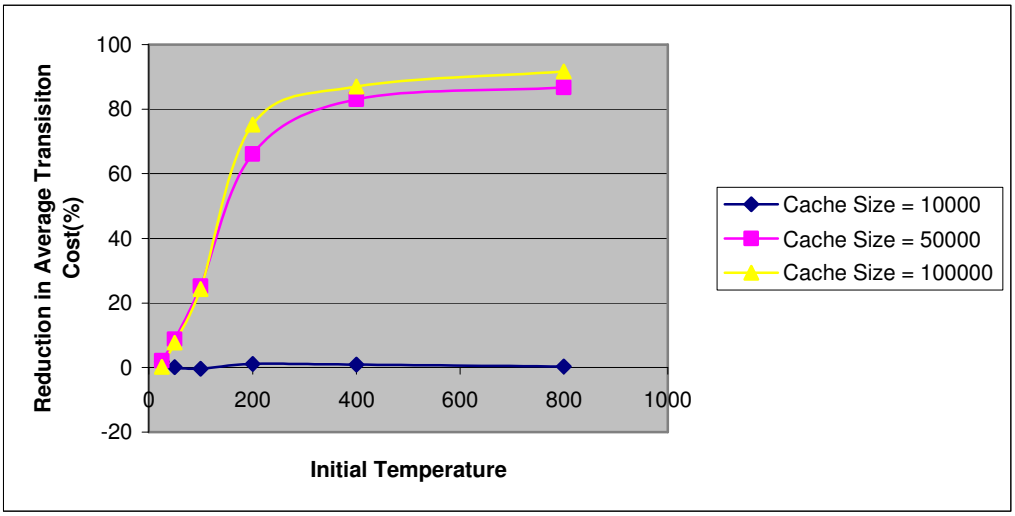

Figure 21. Cost Reduction with changing Cache Size with Heuristic Candidate Generator and Random PS Constructor

For better visualization of the method see Figure 22, Figure 23 and Figure 24. Arrows indicate the transition from gray cell to black cell. Picture on the left side shows the transition without prefetching whereas the one on the right side shows the transition with prefetching. On both sides, white boxes are buildings which are in the destination cells PVS and dark gray boxes are buildings which are in the PVS of the source cell. On the right side, prefetched buildings are filled with light gray. White boxes on the right side indicate the buildings which are not covered by the prefetched set of buildings.

Figure 22 illustrates a transition where the PVS of the destination cell is fully prefetched before the transition happens. Transition has a high probability of occurrence on the average therefore the method favored the transition for prefetching. Figure 23 illustrates a transition with low probability of occurrence. Although the PVS of the destination cell is small, algorithm unfavored the transition and ignored it for prefetching. Figure 24 illustrates a situation which is similar to the first one but with one major difference: the cache size was not sufficient to fully prefetch the destination cell's PVS. Algorithm favored the transition but it was unable to prefetch all buildings.

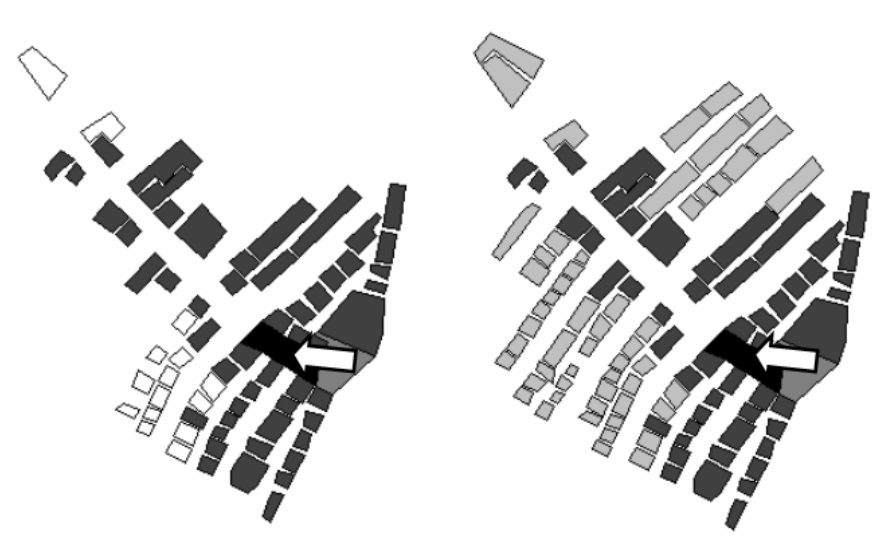

Figure 22. Full coverage of PVS. All buildings in the destination cell's PVS are prefetched.

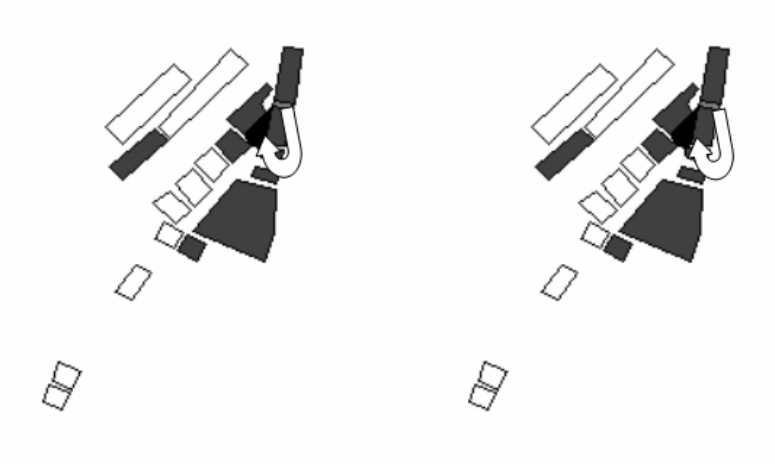

Figure 23. No prefetching for a transition. Nothing is prefetched because the source does not have transitions with high hit count.

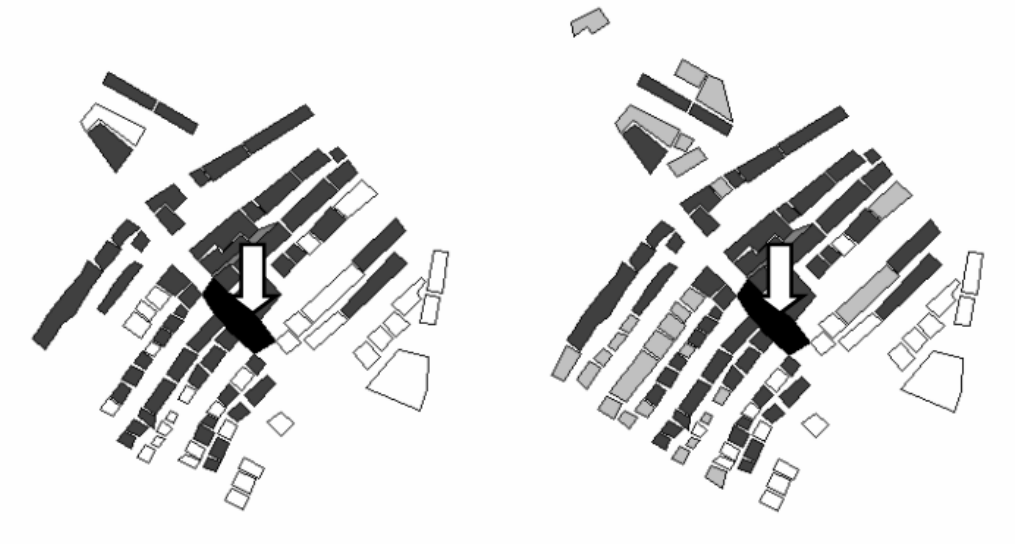

Figure 24. Partial PS coverage. Transition has a high hit count but the cache size is too small for full prefetching of destination cell's PS.

5.2 RESULTS WITH RANDOM CANDIDATE GENERATOR AND RANDOM PS CONSTRUCTOR

The second set of results are collected with RandomCandidateGenerator as candidate generator, ScoringForAverage as scoring policy and AllBuildingsPS-Constructor as prefetching set constructor. See CHAPTER 4 for details of these components. This set represents the performance of the approach without using heuristics for candidate generation.

Table 2. Results with Random Candidate Generator and Random PS Constructor

CACHE SIZE	INITIAL TEMPERATURE	AVERAGE COUNT B.A.	AVERAGE COST B.A.	FIRST SCORE	LAST SCORE	AVERAGE COUNT A.A.	AVERAGE COST A.A.	COST REDUCTION
10000	25	11.99	5913	5066	4991	12.00	5970	-0.96
10000	50	12.05	5930	5066	4778	11.92	5850	1.34
10000	100	11.98	5924	5066	4691	11.55	5813	1.87
10000	200	11.99	5916	5066	4501	11.82	5815	1.70
10000	400	12.08	5918	5066	4263	11.97	5908	0.16
10000	800	11.93	5904	5066	4489	11.87	5865	0.66
50000	25	12.00	5894	5066	4957	11.77	5895	-0.01
50000	50	11.92	5886	5066	4846	11.13	5607	4.74
50000	100	12.15	5949	5066	4696	10.08	5085	14.52
50000	200	12.02	5917	5066	4355	10.28	5093	13.92
50000	400	12.01	5926	5066	4169	10.11	4993	15.74
50000	800	12.07	5933	5066	3839	11.29	5682	4.23
100000	25	12.01	5914	5066	4977	11.77	5936	-0.37

CACHE SIZE	INITIAL TEMPERATURE	AVERAGE COUNT B.A.	AVERAGE COST B.A.	FIRST SCORE	LAST SCORE	AVERAGE COUNT A.A.	AVERAGE COST A.A.	COST REDUCTION
100000	50	12.01	5946	5066	4838	11.04	5588	6.02
100000	100	11.97	5904	5066	4614	10.66	5259	10.92
100000	200	11.99	5928	5066	4597	9.90	5078	14.33
100000	400	11.94	5885	5066	4129	9.32	4757	19.16
100000	800	12.05	5911	5066	3848	11.6	5881	0.50

It is clear that the heuristic was very effective on the performance of the method. Without it the performance decreased drastically as expected. SA performs less than 1/3 of the previous case. Again the pace of reduction decreases after 200 since it becomes probabilistically harder to find better solutions with increasing number of trials. (Figure 25. Performance of SA)

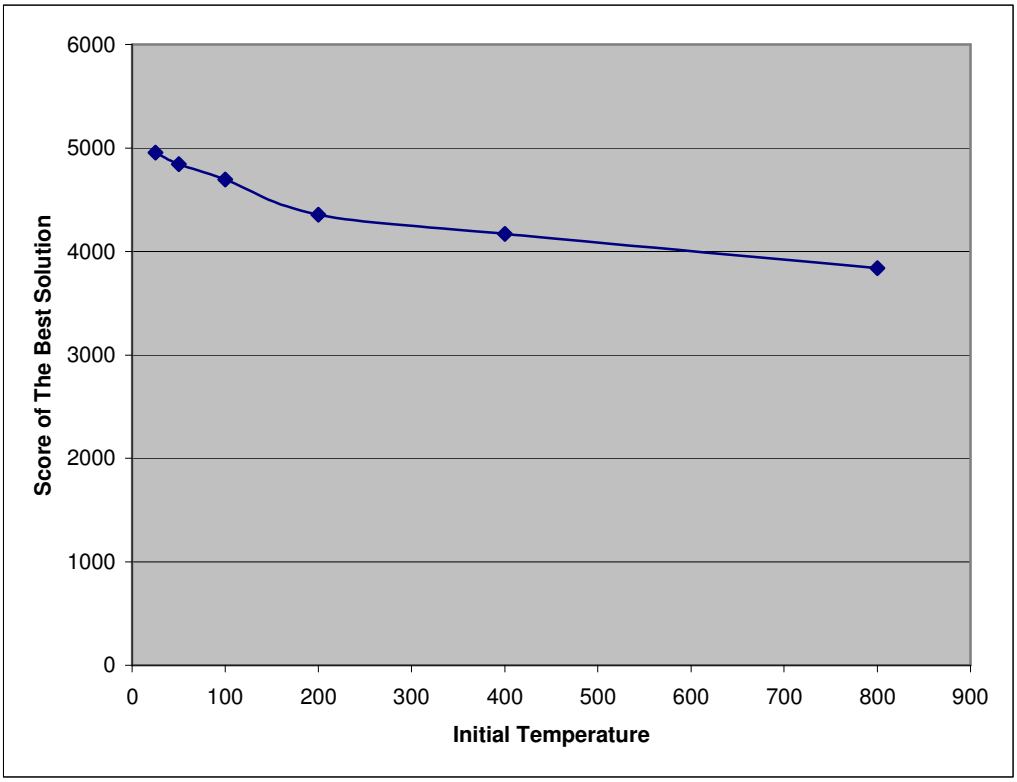

Figure 25. Performance of SA with Random Candidate Generator and Random PS Constructor

The ratio between SA performance and the Cost Reduction performance is again lower than the heuristic supported test series. While the heuristic provides 91% cost reduction for 50% SA optimization, test without the heuristic provides 19% cost reduction for 20% SA optimization. In other words, effectiveness of SA decreases if the heuristic is not utilized. An interesting point to consider is that Cost Reduction decreases for temperatures higher than 400. In the absence of the heuristic, although the SA finds better solutions at these temperatures, Cost Reduction acts in the opposite direction. This shows

that without the guidance of the heuristic during candidate generation, generated candidates may not help to increase transition costs although they seem producing better solutions for SA. In other words the method finds better solutions for global optima but misses the fact that minima on the more probable transitions are the ones to help to improve the results. With the increasing temperature, SA finds those global minima which are not the minima for the probable cases. (Figure 26. Cost Reduction with changing Cache Size)

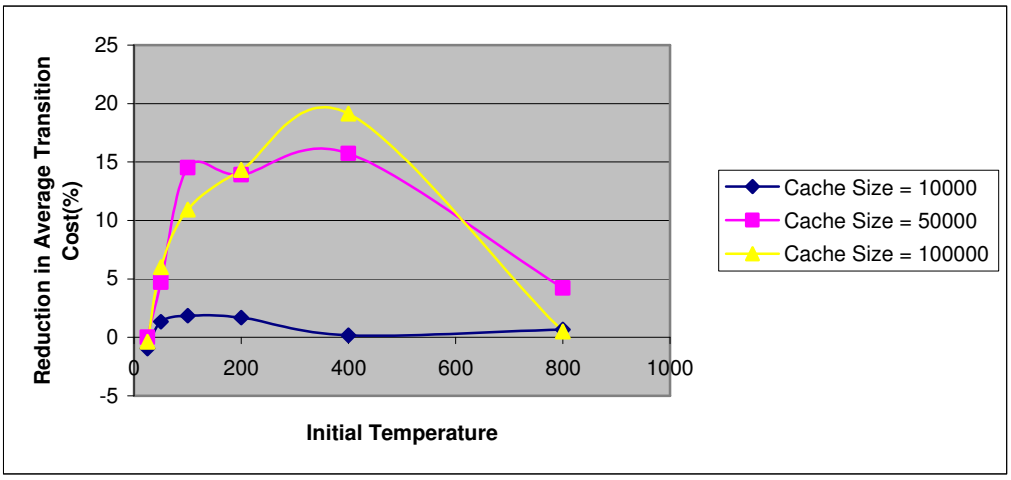

Figure 26. Cost Reduction with changing Cache Size with Random Candidate Generator and Random PS Constructor

5.3 RESULTS WITH HEURISTIC CANDIDATE GENERATOR AND HEURISTIC PS CONSTRUCTOR

Third set of results are collected with CandidateGeneratorForAverage as candidate generator, ScoringForAverage as scoring policy and MostAppearing-BuildingsPSConstructor as prefetching set constructor. See CHAPTER 4 for details of these components.

This set of results is interesting since it shows a heuristic may not improve the cost reduction as much as expected. It is easy to mis-predict the effect of a heuristic. Tested heuristic is injected into the PS construction state of the method. While constructing PS, instead of random order buildings are handled in the decreasing order of their containing nodes where the nodes/view-cells are sorted according to their global hit counts. Buildings in the PVS of more visited cells take place in front of the others in the PS.

Table 3. Results with Heuristic Candidate Generator and Heuristic PS Constructor

CACHE SIZE	INITIAL TEMPERATURE	AVERAGE COUNT B.A.	AVERAGE COST B.A.	FIRST SCORE	LAST SCORE	AVERAGE COUNT A.A.	AVERAGE COST A.A.	COST REDUCTION
10000	25	12.01	5931	5066	4944	11.55	5851	1.34
10000	50	12.02	5916	5066	5063	11.68	5880	0.60

CACHE SIZE	INITIAL TEMPERATURE	AVERAGE COUNT B.A.	AVERAGE COST B.A.	FIRST SCORE	LAST SCORE	AVERAGE COUNT A.A.	AVERAGE COST A.A.	COST REDUCTION
10000	100	12.02	5924	5066	4960	11.70	5800	2.09
10000	200	11.96	5929	5066	4885	11.38	5751	3.00
10000	400	12.01	5895	5066	4878	11.46	5845	0.84
10000	800	11.98	5888	5066	4747	11.81	5840	0.81
50000	25	12.07	5947	5066	4889	11.55	5769	2.99
50000	50	12.04	5918	5066	4840	11.20	5622	5.00
50000	100	12.06	5956	5066	4353	9.73	4962	16.68
50000	200	12.05	5923	5066	4163	8.58	4160	29.76
50000	400	12.03	5911	5066	4064	7.75	3510	40.61
50000	800	12.03	5959	5066	3684	7.87	3265	45.20
100000	25	12.12	5952	5066	4940	11.53	5807	2.43
100000	50	11.95	5910	5066	4795	11.49	5689	3.73
100000	100	11.98	5903	5066	4561	9.19	4683	20.66
100000	200	12.03	5927	5066	3724	5.98	3025	48.96
100000	400	11.96	5912	5066	3757	5.60	2359	60.09
100000	800	11.97	5951	5066	3392	5.40	2098	64.74

Both LAST SCORE and COST REDUCTION columns show results worse than expected. Injection of the heuristic decreased the performance instead of increasing it. When compared with results of the first test series, it is obvious that the heuristic harmful. Heuristic caused maximum cost reduction to decrease from 91% to 65%. After a short inspection, it is seen that selecting buildings from the globally most appearing cells decreased the probability of effective buildings when compared to random selection. Search space is narrowed to those solutions which optimize the graph according to global fetching frequency but for our case, required behavior was optimizing

according to local frequency which is selecting most buildings according to current view-cell. Selecting globally frequently loaded building did not covered frequently loaded neighbor view-cell PVSs. (Figure 27. Performance of SA and Figure 28. Reduction with changing Cache Size)

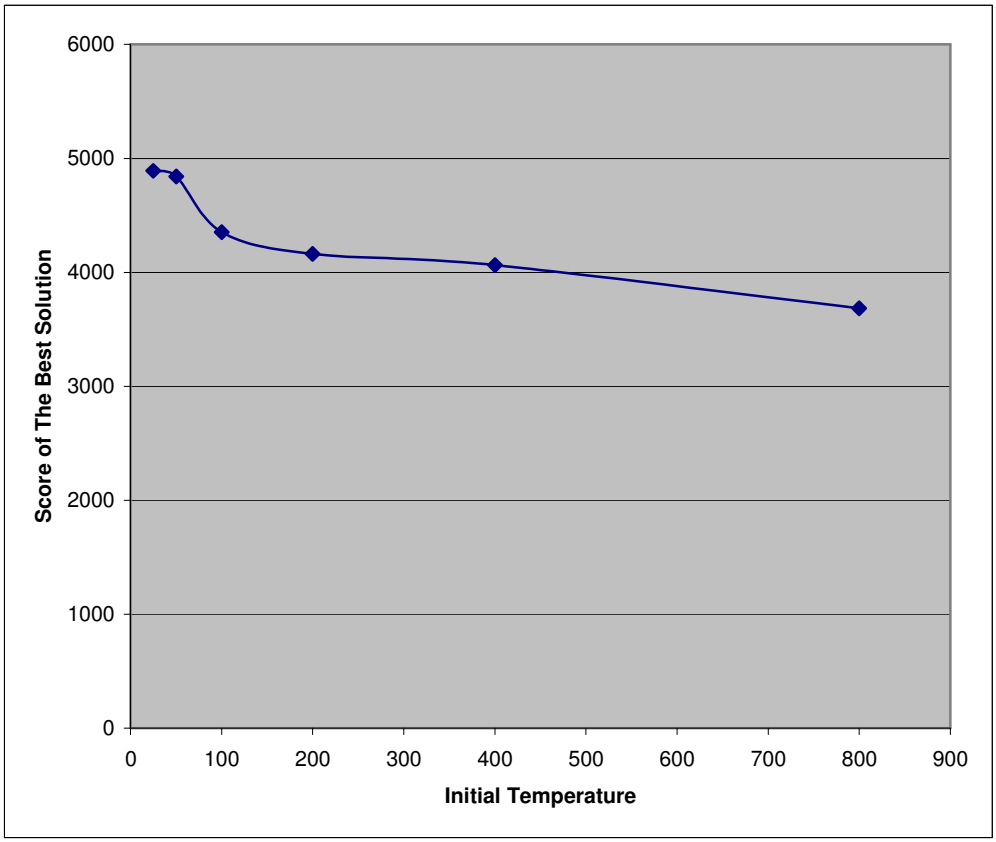

Figure 27. Performance of SA with Heuristic Candidate Generator and Heuristic PS Constructor

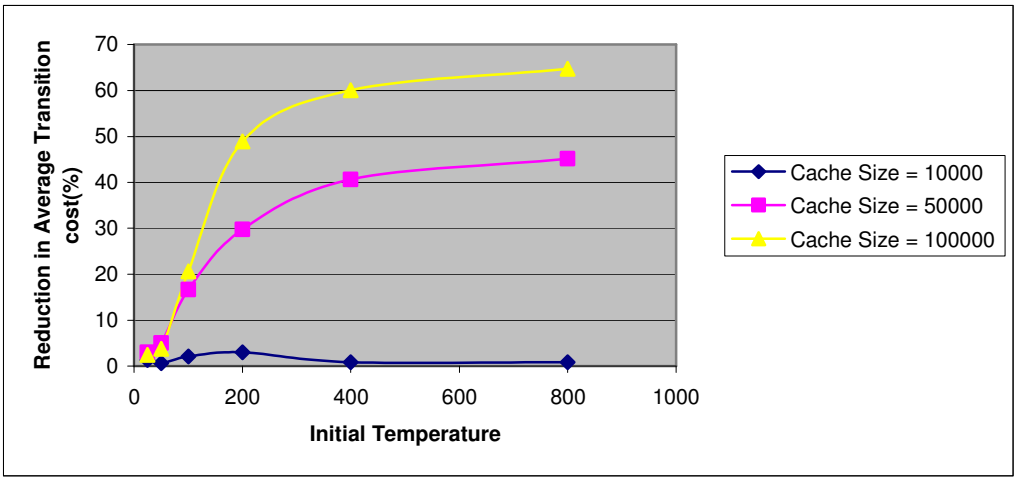

Figure 28. Reduction with changing Cache Size with Heuristic Candidate Generator and Heuristic PS Constructor

5.4 RESULTS WITH TRIVIAL PREFETCHING

Table 4. Results of Trivial Prefetching with varying cache size.illustrates performance of trivial prefetching with different cache sizes. As seen on the table, for smaller cache sizes represented method is more successful than trivial prefetching. As the cache size gets larger clearly trivial method performs better than the represented method.

Table 4. Results of Trivial Prefetching with varying cache size.

CACHE SIZE	AVERAGE COUNT B.A.	AVERAGE COST B.A.	AVERAGE COUNT A.A.	AVERAGE COST A.A.	COST REDUCTION
10000	12.02	5921	11.717	5812	2
30000	12.05	5915	10.84	5336	10
50000	11.99	5913	6.49	3132	47
75000	12.01	5913	1.73	848	86
100000	11.96	5920	0.24	99	98

CHAPTER 6

CONCLUSION AND FUTURE WORK

This study represents a method for implementing prefetching policies and includes a sample implementation with sample heuristics. Method represents the scene as graph where the nodes represent the view-cells and the edges represent the transitions between view-cells. On the defined graph, prefetching is mapped to a graph partitioning problem where the partitions are allowed to intersect. Partitions in other words the groups are interpreted as the prefetching policy. Each node is associated with a set of buildings to prefetch, namely Prefetching Set (PS). PS of a cell is constructed from the buildings contained in the nodes of the groups which contain the cell. A generic probabilistic optimization algorithm, Simulated Annealing, is utilized for producing solutions based on the graph representation. SA provides abstraction and flexibility to inject heuristics to the method.

There are two points where heuristics can be injected into the method. Partitioning/grouping the cells and building the PS of cell-based on these groups. Sample implementation includes two grouping heuristics. First is the simple random grouping. Second heuristic assigns probabilities to transitions with respect to their hit counts. Based on the assigned probabilities transitions

are selected and their source and destination nodes are used to create groups. Second point for injecting heuristics is PS construction stage. There are two implemented heuristics. First one randomly selects buildings among all buildings. Second one selects buildings from the nodes with higher hit counts.

To test the implementation a simulator is developed. Simulator is utilized for both creating the input and testing the implementation. First run creates statistical data necessary for the method to run. Second run does prefetching according to produced results of the method.

Tests show that the method performs up to %95 transition cost reduction compared to non-prefetched environment with selected set of heuristics. It provides focus on the heuristics rather than finding exact solutions. As expected, the effectiveness of the method is strictly bounded to effectiveness of the heuristics used and the allocated cache size used for prefetching. The study claims providing a general approach for the prefetching problem. It is shown that the developed method is successful in fulfilling its claim.

While prefetching it is possible that the viewer stays in the cell more than necessary to complete the prefetching. After completion of the prefetching a second level of prefetching can be utilized for PS of the neighbor cells. Based on this fact developing an improved hierarchical method of prefetching will produce better results.

Graph representation enables utilization of level of detail (LOD) methods. Different LODs can be prefetched to provide better performance. Graph abstraction can be utilized to make decisions about LOD of the building to prefetch.

Hierarchical approach and LOD capabilities are considered as the next step of this method with developing more effective heuristics for group forming and PS construction.

REFERENCES

[1] T. Möller and E. Haines: *Real-Time Rendering, 2nd edition*, A. K. Peters, 2002. Cited on page 1, 2, 22, 46, ix, xxiii.

[2] Seth J. Teller and Carlo H. Sequin: *Visibility preprocessing for interactive walkthroughs*, Computer Graphics (Proceedings of SIGGRAPH 91), 25(4):61–69, July 1991.

[3] John Airey: *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.

[4] 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.

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

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

[7] 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.

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

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

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

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

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

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

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

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

[16] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario.: *Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes*, Computer Graphics Forum, 17(3):243–254, 1998.

[17] Carlos Saona-Vazquez, Isabel Navazo, and Pere Brunet.: *The visibility octree: A data structure for 3d navigation*, Computer & Graphics, 23(5):635–644, 1999.

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

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

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

[21] 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.

[22] Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. Rendering Techniques 2000: 11th Eurographics Workshop on Rendering, pages 71–82, June 2000. ISBN 3-211- 83535-0.

[23] Eyal Teler, Dani Lischinski: *Streaming of Complex 3D Scenes for Remote Walkthroughs*, Computer Graphics Forum, 20(3), pages 17-25, September, 2001.

[24] Christopher Zach, Konrad Karner: *Prefetching Policies for RemoteWalkthroughs*, Proceedings of WSCG 2002, February, 2002.

[25] Vladlen Koltun, Yiorgos Chrysanthou, Daniel Cohen-Or: *Hardware-accelerated from-region visibility using a dual ray space*, EGWR01:12th Eurographics workshop on Rendering, pages 204-214, June, 2001.

[26] Zhi Zheng, Tony K. Y. Chan: *Optimized Neighbour Prefetch and Cache for Client-server Based Walkthrough*, 2003 International Conference on Cyberworlds December 03 - 05, 2003. p. 143

[27] S Kirkpatrick, CD Gelati Jr, MP Vecchi: *Biology and Computation: A Physicist's Choice*, 1994

APPENDIX A - IMPLEMENTATION DETAILS

This section gives details at the level of actual classes. Significant amount of the information stated here can be found in Chapter IV with a different point of view. If the reader is not specifically interested in class level, it is not necessary to cover this section.

Design and implementation of the optimization algorithm has three main components.

1. Generic Simulated Annealing Implementation
2. Graph Implementation
3. Adaptation of Simulated Annealing for Prefetching optimization

1 GENERIC SIMULATED ANNEALING

This component consists of five interfaces and a generic Simulated Annealing implementation:

1. ICoolingSchedule
2. IScorePolicy
3. ISolution
4. ISolutionSpace
5. IStateTransition
6. SimulatedAnnealing

Pseudo code of the generic SimulatedAnnealing implementation is given below. Before initiating the flow implementations of interfaces (interface 1, 2, 4 and 5) are registered to SimulatedAnnealing instance.

```
ICoolingSchedule        coolingSchedule;
ISolutionSpace          solutionSpace;
IStateTransition        stateTransition;
IScorePolicy            scorePolicy;
ISolution               bestSolution;
ISolution               candidate;

while(coolingSchedule.getTemperature() > 0) {
        candidate = solutionSpace.getCandidate();

        if(candidate better than bestSolution) {
                bestSolution = candidate;
        }

        candidateScore = scoringPolicy.score(candidate);
        currentScore =
scoringPolicy.score(solutionSpace.getCurrent());
        temperature = coolingSchedule.getTemperature();

        if(stateTransition.checkTransition(temperature, currentScore,
        candidateScore)) {
                solutionSpace.acceptCandidate();
        }

        coolingSchedule.cool();
}
```

Pseudo code given above summarizes the flow of the algorithm and the interfaces registered.

ICoolingSchedule **:** Handles cooling. With each call to *cool()* temperature kept inside is decreased.

ISolutionSpace **:** Responsible for candidate generation.

ISolution **:** Represents solutions in the solution space.

IStateTransition **:** Decides whether candidate solution should be accepted or rejected as the current solution.

IScorePolicy **:** Responsible for evaluating the solutions.

2 GRAPH

Graph component contains 5 classes that represent a graph.

Building **:** Building has a cost associated with it.

Node **:** Node has a set of buildings as its PVS, a set of transitions for both incoming and outgoing types, a list of groups that the node belongs to.

Transition **:** Transition has fields, source node, destination node and a list of buildings to be loaded to complete the PVS of destination node.

Graph **:** Graph is a container holding a list of nodes, a list of transitions and a list of groups.

Group **:** Group contains the list of nodes forming it with a set of buildings forming the PS.

3 ADAPTATION

Generally, adaptation of Simulated Annealing is performed via implementation interfaces of generic implementation. Auxiliary classes cover the distance between SA component and graph component. For performance constraints instead of evolving solution space back and forth, an auxiliary class SolutionSpaceMove is utilized with undo capability. Candidate generation is separated from SolutionSpace for modularity. ISolutionSpace, IStateTransition, IScoringPolicy, ICoolingSchedule and ISolution is implemented on these auxiliary classes. Depending on these facts current design is as follows:

3.1 UTILS

This class provides util functions for set operations on lists of buildings.

3.2 SOLUTION

This class represents a specific partitioning of the graph. In fact there is only one solution and SolutionSpaceMove instance to produce a new solution on the single instance of Solution. This approach is used for increasing performance.

3.3 SOLUTION SPACE MOVE

This class represents the necessary action defined on the Solution to evolve it to a new one. A SolutionSpaceMove can be an add action or a remove action. The add action adds a node to a group where remove removes a node from a group. SolutionSpaceMove has the unique property of undoing itself when StateTransition rejects moving to candidate solution.

3.4 CANDIDATE GENERATOR

Candidate generator generates a SolutionSpaceMove instance for creating a candidate partitioning based on current partitioning. First step of candidate generation is randomly selecting a node to operate on. After the node is selected, type of the move is decided, add or remove. According to the defined type CandidateGenerator tries to find a group to add the node into or a group to remove the node from. If this search fails, it is performed for the other type. If no suitable groups are found, whole cycle is done again for another randomly selected node until a suitable move is found. CandidateGenerationHeuristic instances are registered to CandidateGenerator to inject the defined heuristics in the candidate generation process.

DefaultCandidateGenerator :

Decides to add a node to a group or remove a node from a group with probability of 50-50. To add a node to a group, a Node and a neighbor Node are randomly selected. From the groups of the neighbor node selects one group randomly. To remove a Node from a group, a Node and a groups among the Node's groups are randomly selected.

CandidateGeneratorForAverage :

Decides to add a node to a group or remove a node from a group with probability of 50-50. To add a Node to a group, transitions with higher hit counts are favored whereas to remove a Node from a group transitions with lower hit counts are favored. When a transition is selected its source and destination Nodes are used to produce a candidate.

3.5 PS CONSTRUCTOR

AllBuildingsPSConstructor **:** All buildings contained in the groups of to Node are added to PS of the Node in random order.

MostAppearingPSConstructor **:** Buildings with higher hit count, in other words buildings contained in the PVS of the Nodes with higher hit count are inserted first into the PS.

3.6 CONSTRAINTCHECKER

Instances of this class are registered to candidate generator for narrowing the search space. When a move is generated it is checked against the constraints. It it fails the test a new candidate is generated and tested. This continues until a suitable candidate is generated.

Current implementation extends ConstraintChecker to MemoryConstraint-Checker. If a candidate exceeds memory limit with the constructed PS, the candidate is discarded.

3.7 SOLUTION SPACE

SolutionPace wraps instances of CandidateGenerator, SolutionSpaceMove and Solution. By utilizing these instances it provides function necessary for generic Simulated Annealing implementation.

3.8 COOLING SCHEDULE

Cooling schedule in current implementation is an integer with is decreased by one at each iteration.

3.9 STATE TRANSITION

State transition function $P(e,e',T)$ is defined as 1 if $e' < e$, and $\exp((e - e') / T)$ where e and e` are the scores assigned with the associated Scoring instance and T is the current temperature. This allows moving to worse states depending on T and the difference between energies of the candidate state and the current state. Probability of moving to a state with higher energy (worse state since we are trying to minimize the energy) increases with increasing T and decreasing difference in energies of two states. On the other hand P is always 1 when

candidate state has a lower energy. This property of SA avoids being stuck at a local minimum by allowing it to jump out of a local minimum to a state (probably with higher energy when T is high) which can lead to global minima.

3.10 SCORING

Scoring implements IScorePolicy and extended with three specific scoring schemas: ScoringForMaximum, ScoringForAverage and ScoringForPopular-Path.

ScoringForAverage **:** *Average scoring policy calculates the average cost of transitions in the graph. The straightforward formula is* Σ *cost(t) / number of t.*