Prefetching Optimization for Multiplayer Online Games Using Simulated Annealing

Safak Burak Cevikbas, Mert Degirmenci, Veysi Isler

Middle East Technical University, Department of Computer Engineering

\{ safak.cevikbas, mert.degirmenci, isler \} @ ceng.metu.edu.tr

Abstract

We propose a prefetching method to improve smoothness of client side rendering in distributed virtual urban environments. In order to find optimum prefetching policy, this method reduces the prefetching problem to a graph partitioning problem. The scene is represented by a graph where the vertices denote view-cells and the edges denote the adjacencies among the cells. We utilize the Simulated Annealing algorithm to find an approximation for the optimum partitioning. The proposed method aims to minimize view-cell transition cost by utilizing statistical data obtained from clients. Multi-agent client simulator has been implemented for testing the efficiency gain and detecting characteristic information about clients. Our algorithm iteratively refines the prefetching policy by using client tendency data obtained from the simulator. Experimental results confirmed significant improvements on client rendering of virtual urban environments.

Keywords: Visibility, Prefetching, Simulated Annealing, Urban Environments

1. Introduction

As larger data sets become available for distributed urban environments, the importance of visibility increases. Increasing complexity of models and the need for interactive walkthroughs in them also lead visibility studies to become a significant research area. A visibility method aims to compute a photo-realistic image by assuring that the primitives behind do not incorrectly occlude the front ones. Common approach is to let the graphics hardware estimate the visible primitives after traversing all primitives in the
scene. Current graphics hardware can render all the primitives in any order and estimate the visible primitives by utilizing z-buffer. Z-buffer keeps the depth of the closest rendered primitive on a pixel basis and culls the primitives behind.

Sending all primitives to graphics hardware is an inefficient method for large scenes such as urban environments where a few close buildings might occlude hundreds of others. Processing all occluded buildings in the remaining steps of the rendering pipeline is unnecessarily time consuming and useless.

Visibility researches focus on estimation of visible primitives by culling the invisible primitives in the scene. Visibility culling approaches are classified as view-frustum culling, back-face culling and occlusion culling. The purpose of view-frustum culling is to discard the primitives outside the view frustum, keeping the focus on primitives observable by camera. Back-face culling, on the other hand, avoids rendering the geometries facing away from the viewer. Although these methods increase efficancy of rendering pipeline, both of them do not eliminate obscured or occluded primitives inside the view frustum as seen in Figure 1.

Occlusion is inevitable in complex scenes like urban environments. Visibility studies focus on exploiting this fact. The aim of visibility culling methods is to obtain exact visible set. However, real time computation of a visible set is infeasible for interactive applications. Hence, conservative visibility methods, which overestimate the visible set of primitives, are proposed to compromise between culling accuracy and culling speed. This overestimated visible set of primitives is referred to as Potential Visible Set (PVS). PVS includes all visible primitives plus relatively smaller group of invisible primitives. In other words, an occluded object might be estimated as visible, however a visible object is never treated as occluded in conservative visibility.

Solutions for occlusion culling can be categorized into two major groups. These are point-based and region or cell-based approaches. While point-based methods calculate PVS for each location of the viewer at run-time, region-based methods generally require offline calculation of PVS for a specific region.

Region-based visibility methods partition the scene into sub-regions which are called view-cells. Potential visible set of each view-cell is computed in preprocessing stage. When the viewer is in a view-cell, it is sufficient to render only the objects in the corresponding PVS without visual loss. PVS is used for rendering as long as the viewer does not leave the view-cell associated with it. However, this approach may cause stalls during view-cell transitions.
since a bulk of data might be transferred from the server to the clients when view-cell transition occurs.

In this article, we propose a method to avoid potential stalls during view-cell transitions. A well-known approach to avoid stalls is called prefetching which tries to predict the next transition and the data to be loaded. Thus, predicted data for the next view cell is transferred to the client before the transition occurs. The proposed method is based on the graph representation of view-cells and the transitions among them. We have used a stochastic algorithm, namely Simulated Annealing, to find the optimum prefetching policy. In our method, prefetching problem is transformed into a graph partitioning problem. A multi-agent client simulator has also been implemented to test and refine our prefetching policy progressively. The statistics obtained from clients have been fed into heuristics used in simulated annealing.

Section 2 reviews the related work on visibility and prefetching. Section 3 describes the proposed method. Section 4 discusses the results. Finally, section 5 concludes the study and describes the future work.

2. Related Work

Required frame rate for an interactive walkthrough application is at least 20 Hz [7]. There always have been scenes more complex than the best performing hardware can handle with brute force. Even if the scene to be
rendered is not complex, a viewer may require large data from the network upon a position change. Rather than traversing all primitives to the graphics hardware, visibility methods enable culling the invisible primitives and processing less number of primitives at the early stages of rendering pipeline. Thus, we can utilize the occlusion culling algorithms to increase the frame rate.

The performance gain of a visibility culling method increases when it filters invisible geometry out as early as possible in the rendering pipeline. Rather than processing each polygon, objects in the scene are hierarchically modeled to cull a group of geometry. Visibility methods increase the performance by utilizing this scene hierarchy.

Due to occlusion large number of invisible objects may remain, after utilizing view frustum and back-face culling approaches. Especially in distributed virtual environments, where the number of objects is large, occlusions can be turned into an advantage. Therefore, precomputation of potential visible sets are crucial for distributed virtual environments.

Culling methods are classified with respect to accuracy as exact, conservative, approximate or aggressive. The target of visibility estimation is to obtain the exact visible set. However, exact visible set computation is extremely costly for real-time complex virtual environment applications. Therefore, conservative visibility algorithms which overestimate the visible primitives are preferred when the accuracy of visibility can be traded [8, 9].

Approximate algorithms might overestimate or underestimate the visible
primitives without guarantee of visual accuracy. Aggressive methods, on the other hand, generally underestimate the number of visible primitives, and gain efficiency in rendering pipeline by trading preciseness of scene. Aggressive and approximate visibility methods risk culling visible geometry for the sake of rendering performance.

One of the significant classifications has to be highlighted: online visibility culling and offline visibility culling. While the former determines the visibility of a scene geometry on the fly, the latter computes the visibility at the preprocessing stage [10].

Online visibility culling algorithms re-compute PVS at each movement of viewer, i.e. changing the viewpoint or view direction. Methods computing the PVS on each movement of the client are classified as point-based visibility culling methods [10]. Some algorithms amortize the cost of PVS computation by computing visibility only 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 PVSs [11,12].

Offline visibility culling methods generally partition the scene into cells and precompute the PVS of each cell. 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 is 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 PVS, though an approximate PVS may cause undesirable flickering. Sampling approach can not estimate PVS conservatively since lots of objects may be seen from a small hole. Moreover, region based visibility is not effective with occluders smaller than the cell since such an occluder generates only a finite conservative shadow frustum behind it as seen in Figure 3 [10, 22].

Region based visibility methods are more suitable for server-client applications to visualize large data sets representing urban environments. However, they have two drawbacks: high pre-processing time and high storage requirements. Moreover, a transition from one view-cell to another may require loading bulk of data in order to build PVS of the destination view-cell. While loading bulk data, either the rendering pauses or the scene is rendered with incomplete data. On the other hand, region based methods allow utilization of prefetching strategies. While a viewer is walking through a large urban environment which is decomposed into view cells, prefetching the PVS
of destination cell before leaving the current one overcomes the stall during cell transition and enables smooth rendering. The prefetching methods covered in this section are considering potential visibility of geometric data.

Both Teller [7] and Zach [8] fetch the scene data in small chunks of different level of detail (LOD) representations. Scene data to be retrieved is selected and LOD is estimated based on a benefit/cost evaluation. They make use of a short history of viewer motion to predict the future moves according to viewer direction and position. Based on the prediction performed, a certain scene data is prefetched for possible rendering.

Koltun utilizes a simple prefetching method that considers a vicinity of current view-cell [10]. In this method, PVSs of all adjacent cells are prefetched together with the PVS of the current cell. When the viewer passes to an adjacent cell, PVS of the new cell is already fetched and rendering continues smoothly. However, this method redundantly fetches the PVSs which are already loaded in the previous transition.

Zheng and Chan [9] further improved the method by deferring prefetching until the viewer tends to leave the current view-cell. They decrease the number of cells to be prefetched by selecting only a subset of neighbor cells according to the tendency of the viewer.

Methods mentioned above, do not track viewer tendency and do not consider global optimization except a short history or a limited proximity. The proposed method in this paper provides flexibility by considering character-
istics of an urban environment.

3. Prefetching Optimization

The proposed method is developed for walkthrough applications in very large urban environments. A typical urban scene can be considered as combination of roads and buildings. Roads are the parts of the scene where a viewer can walk on. Buildings are the objects to be rendered and they are the main source of occlusion. A viewer navigating in the roads of an urban environment may only see a small part of the scene since the close buildings occlude those behind.

To alleviate network and rendering burden of occluded parts in a virtual urban environment, our prefetching optimization algorithm first creates graph representation of the scene. Multi-agent simulator is then used to collect statistics about the virtual urban environment. After we employ the simulated annealing to optimize prefetching, we run the simulator again to collect feedbacks. Details of our algorithm is described below.

1. Create Graph Representation of the Scene
2. Collect statistics of a client walking through the scene using a simulator
3. Optimize the prefetching by partitioning the graph using simulated Annealing
4. Run the simulator again to obtain reports of potential hazard or benefit in our prefetching policy.
5. Feed the reports to our optimization heuristics and go to step 3.

3.1. Graph Representation

Our method is built upon the graph representation of the scene. We define the view-cell graph as \( <N, E> \). \( N \) is a set of view-cells. Each cell has an associated set of buildings to be rendered when the viewer is in that view-cell. \( E \) is a set of transitions. A transition is defined as \( <S, D, L> \) where \( S \) is the source view-cell, \( D \) is the destination view-cell and \( L \) is the set of buildings which are in the PVS of \( D \) but not in the PVS of \( S \).

When a transition from one node to another occurs, it is sufficient to load buildings in \( L \) to properly built PVS of the destination cell. Buildings which are not necessary for \( D \) are discarded. The process of discarding is assumed to have no cost. Therefore, the transition cost equals to the cost of buildings which are loaded for the destination view-cell.
3.2. Collecting statistics

Most of the characteristics of a virtual urban environments, such as popularity of view-cells or paths, are determined by it’s users. A prefetching optimization algorithm should benefit from user statistics to get the upmost performance.

In our implementation, we have developed a multi-agent simulator that models clients of a virtual environment. This simulator collects the statistics of users and feeds this information to our prefetching algorithm. We can also observe relative reduction in the network communication by inspecting successive runs of our simulator.

3.3. Prefetching as Graph Partitioning

Basic application of prefetching would be loading PVSs of neighbor cells [4]. This approach is static in terms of customization and prioritization. It ignores scene specific properties like the time spent in cells and the cell size. When the adjacent cells have small dimensions or the viewer stays for short time in a view-cell, it may not be possible to prefetch all neighbouring PVSs properly which may cause a stall for the subsequent transition. A better solution should consider scene specific informations such as viewer trends and local memory limitations of the client computers.

Instead of defining fixed rules of prefetching, we propose a more flexible method based on the representation of the scene as a graph. We partition
the graph into groups of adjacent nodes and construct sets of buildings for
each of these groups. We name these buildings sets as Prefetching Sets (PS).

For the simplicity, we loosen the definition to allow partitions to overlap.
Figure 2 illustrates a sample partitioning. When the viewer is in a node
which is included in more than one group, the PS is constructed from union
of building sets with the help of heuristics. It can be shown that the same
approach works for non-overlapping partitions.

We utilize two basic heuristics for grouping nodes:

- High probability transitions pull the source and the destination nodes
  into the same group.
- Nodes in which the viewer stays for short time pull their adjacent nodes
  into the same group.

For constructing PSs, we define the following methods:

- Randomly generated subset of PVS.
- Frequently appearing buildings.

Figure [6] illustrates the cost reduction caused by prefetching. First image
shows a transition without prefetching and the second shows the same trans-
section with prefetching. Arrow shows the view-cell transition. White boxes
on the first image are the buildings in the PVS of destination cell. Dark
gray boxes are the buildings in the PVS of the source cell. Prefetched build-
ings are shown with light gray on the second image. It can be seen that
the prefetched set is a superset of the destinations cell’s PVS therefore the transition cost becomes zero.

However it is not always possible to cover whole PVS of the destination cell [7], PS constructor becomes important for such cases. Thus, it tries to build the most effective PS which fits best in the available resources.

3.4. Optimization by Simulated Annealing

Simulated annealing (SA) is a generic probabilistic meta-algorithm regarding to the global optimization problem. It locates a good approximation to the global optimum of a given function in a large search space [6]. Simulated annealing approach builds an analogy between physical systems. A point $s$ in the search space is analogous to a state of the system and $E(s)$ is analogous to the internal energy of the system at the current state. The
goal is to bring the system from an arbitrary initial state to a state with the minimum energy. Basically SA repeats in five steps as iterations until the computation budget is exhausted. Computation budget is realized with a global variable called temperature. The five iterating steps of SA are summarized as follows:

1. In the solution space, choose an adjacent solution $s'$ of the current solution $s$.
2. Calculate the energy required to reach $s'$ from $s$ as $E(s')$.
3. Store $s'$ if it is the best solution so far. i.e. $E(s')$ is the minimum so far.
4. Decide moving to $s'$ or staying at $s$ using Temperature information.
5. Decrease Temperature.

For the partitioning problem, the search space is defined as the set of all possible partitioning of view-cell graph since each partitioning of the view-cell graph is a solution.

In the defined search space, an adjacent solution of current solution can be reached by one of the following:

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

Energy function to be minimized by SA is defined as $E(s) = C(s)$ where $C(s)$ is the average of view-cell transition costs.

Last three steps of the SA are the same for solving graph partitioning problem. What separates SA from greedy algorithms is the function used for step 4. Decision is made by a probability function of acceptance $P(E(s), E(s'), T)$ where $T$ is temperature. $P$ allows moving to worse solutions (solutions with higher energy) depending on $T$ and the energy difference between the candidate solution and the current solution. Moving to an unobserved solution becomes more probable increasing $T$ and decreasing energy difference between the solutions. On the other hand, $P$ is always 1 when candidate state has a lower energy (better solution).

To optimize prefetching of scene data based on probabilistic behaviors of clients, SA algorithm has been applied repeatedly exploiting the statistics obtained from our multi-agent simulator. For a virtual environment application with enough clients, it is trivial to gather client’s probabilistic behavior.
Figure 8: (a) View-cell graph of virtual urban environment with red markers placed to indicate monsters (b) client statistics obtained from our multi-agent simulator

Server can record the statistics about the paths and the transitions on each client request. In our implementation, we have generated client tendency on specific view-cells using an artificial scenario of moving monsters. In this scenario, clients that share the same view-cell with the monsters tend to spend less time on that view-cell. As illustrated in Figure 8, we can infer the characteristic information about client tendency after building an artificial model of clients. Depending on the application of virtual urban environments, similar models can be used to test the prefetching policy.

Naturally, we have observed less client durations on view cells where the monsters wander. This statistical information is extremely significant in order to reduce the transition overhead. By iteratively exploiting client durations on view cells, we can capture frequently used objects and adjust PSs accordingly. In the following section we analyze the efficiency gained using this information.

4. Results and Discussion

Test runs are performed on a test-bed composed of 100 tourists, 93 nodes and 135 buildings. Buildings and roads are extracted from satellite images of Ankara city. Input statistics are collected with 1000 iterations of our multi-agent simulator. Each tourist makes a single view-cell transition in each turn. Therefore, there are $100 \times 1000 = 100000$ view-cell transitions in total.
Figure 9 shows a sample view from our experiments. Two clients with different prefetching policies is shown in the figure. Both clients have performed the same transitions. However, the client on the left employed our adaptation of PS and gained better performance.

Each building is assigned to a cost which is proportional to the area covered by it. This cost corresponds to the memory occupied by a building after it is fetched. Cache size is the limit defined for fetching.

In order to collect statistics from a distributed urban environment, our implementation of a multi-agent simulator repeatedly tests the prefetching method used. To obtain maximum gain from our prefetching policy, results of the simulator have been fed into optimization algorithm. For most practical purposes, clients of a virtual urban environment can be tracked to collect statistics about frequently used paths. These statistics are required to employ heuristics for our optimization algorithm.

Figure 10 shows the results of exploiting statistical feedbacks to our algorithm. On each iteration, client simulator collects statistics and outputs efficiency gained by our optimization. Those statistics are then fed into our implementation of Simulated Annealing algorithm to progressively refine prefetching policy employed. As it can be observed from the figure, probabilistic behavior of clients can be exploited to lower network communication when statistical input is supplied.

Following paragraphs show results of three different combinations of candidate PS generator. For each combination, two charts are presented. First chart shows the results in terms of defined scoring policy which is the average view-cell transition cost.

Second chart shows the reduction in average transition cost when the partitioning schema is utilized on the test scenario. In other words the first and second charts show the theoretical and the actual reduction respectively.
First set of results are collected using heuristics based candidate generator and random PS constructor. Results show that SA implementation increases its effectiveness with increasing initial temperature as expected. However, energy decrease slows as finding better solutions becomes harder for initial temperatures higher than 200. In other words, score converges to some lower bound (Figure 11). Cost reduction shows more dramatic results. 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 illustrated in results, while reduction is near zero with low cache sizes it can hit to 96% with higher cache sizes (Figure 12).

Second set of results are obtained using random candidate generator and random prefetching set construction. This set represents the performance of the approach without using heuristics for candidate generation. It is clear that the heuristics are very effective on the performance of the method. Without usage of them, the performance decreased drastically. 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 13). 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 optimization provided by SA decreases
Figure 11: Simulated Annealing performance with heuristic candidate generator, scoring policy for average cost minimization and random prefetching set constructor

Figure 12: Cost Reduction with heuristic candidate generator, scoring policy for average cost minimization and random prefetching set constructor
Figure 13: Simulated Annealing performance with random candidate generator, average cost reduction as scoring policy and random prefetching set construction

when heuristic is not used. 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 (Figure 14). This shows that without the guidance of the heuristic during candidate generation, created candidates may not help decreasing the transition costs although they seem producing better solutions for SA. In other words the method finds better solutions for global optima but misses the minimum on the more probable transitions.

Third set of results are collected with heuristic candidate generator and most appearing buildings statistics are used in prefetching set constructor. This set of results is interesting since it shows a heuristic may not improve the cost reduction as expected. It is easy to mispredict the effect of a heuristic. Tested heuristic is injected into the PS construction state of the method. While constructing PS, instead of random order construction, buildings are handled in the decreasing order of global hit counts.

Buildings in the PVS of more visited cells take place in front of the others in the PS. SA performance and cost reduction show results worse than expected (Figure 15 and 16). 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 has not produced an improvement.

In previous experiment, employed heuristic caused maximum cost reduction to decrease from 91% to 65%. After an inspection of our multi-agent
simulator, it is observed that selecting buildings from the globally most appearing cells decreased the probability of prefetching the buildings that are distributed well over the virtual environment. Search space is narrowed to those solutions which optimize the graph according to global fetching frequency but for our case, required behavior was optimizing the local frequency namely, selecting most viewed buildings according to current view-cell.

5. Conclusion and Future Work

This study proposes a method for implementing prefetching policies and includes sample application of the method. Method represents the scene as graph on which the prefetching is mapped to a graph partitioning problem. A generic probabilistic optimization algorithm, Simulated Annealing, is utilized for producing solutions based on the graph representation.

To test the implementation, a simulator is developed. Simulator is utilized for both creating the input statistics 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, and reports the results.

Results show that the method changes the focus on the heuristics from finding exact solutions. As expected, the effectiveness of the method is
Figure 15: Simulated Annealing performance with heuristic candidate generator, scoring policy for average cost generation and most appearing buildings heuristic as prefetching set constructor.

Figure 16: Cost reduction with heuristic candidate generator, scoring policy for average cost generation and most appearing buildings heuristic as prefetching set constructor.
strictly bounded to effectiveness of the heuristics used and the allocated

cache size used for prefetching. The study claims providing a general ap-

proach for the prefetching problem. It is shown that the developed method

is successful in fulfilling its claim.

However, the method has drawbacks to be further investigated. It is not

always easy to develop an efficient heuristic when SA’s tendency to clus-

ter solutions around global optima does not comply with heuristic employed.
Moreover, method only converges to an approximate prefetching policy rather

than an exact solution. Utilization of the method requires careful manage-

ment of input statistics to avoid out-dated input which can reduce the effec-

tiveness of the method.

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 study for developing more effective heuristics on group forming

and PS construction.

References

[1] Eyal Teler, Dani Lischinski: Streaming of Complex 3D Scenes for Re-

mote Walkthroughs, Computer Graphics Forum, 20(3), pages 17-25,


for Client-server Based Walkthrough, 2003 International Conference on

Cyberworlds December 03 - 05, 2003. p. 143


