Analyzing Pre-fetching in Large-scale Visual Simulation

Chu-Ming Ng, Cam-Thach Nguyen, Dinh-Nguyen Tran, Tiow-Seng Tan* Shin-We Yeow†
National University of Singapore G Element Pte Ltd

ABSTRACT

This paper studies pre-fetching mechanisms that support walkthroughs of virtual environments with overall data sizes exceeding current core memory capacities of PCs. Specifically, it formulates the pre-fetch problem using motion characteristics of the viewer, the data density of the virtual environment, and the data transfer capability of the disk. It further analyzes two fundamental classes of pre-fetching schemes, which are classified by their uses of certain pre-assigned spatial or temporal thresholds. The analysis is supported and verified by experiments on terrain walkthroughs performed on different configurations of PCs. An interesting finding is the understanding of how the terrain data density supportable is a function of the duration of time allocated to disk-level pre-fetching. On the whole, this paper provides a methodology for analyzing pre-fetch performance as well as presents insights instrumental in the design and optimization of out-of-core walkthrough systems.

CR Categories: I.3.5 [Computer Graphics]: Methodologies and Techniques; I.3.6 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality

Keywords: virtual environment, terrain, walkthrough, out-of-core algorithms

1 INTRODUCTION

The visual simulation of virtual environments plays an important role in a variety of applications such as training and simulation in large scale military war games, verification and visualization of complex computer-aided-design models, and the walkthrough or flythrough of an architectural building or an urban environment. Large-scale virtual environments typically have overall data sizes that exceed current core memory capacities. Due to the enormous amount of data involved, traditional rendering algorithms need to be extended to deal with the fact that the entire dataset is too large to fit in core memory. In addition, these algorithms are designed to work on PCs as they are attractive alternatives to high-end specialized graphics machines due to the PC's rapidly increasing performance in handling graphics at low cost.

The problem of out-of-core visualization is addressed on several fronts, which we broadly classify into three closely interlinked sub-problems that require algorithmic innovation. These sub-problems are *render*, *query* and *pre-fetch*, which correspond, respectively, to processing (retrieved) data to be displayed, identifying and organizing data to be retrieved, and retrieving data into core memory in anticipation of their needs for processing in the near future.

The render sub-problem arises from the complexity of out-of-core data which makes it infeasible to render the geometry of the entire scene. State-of-the-art approaches focus on reducing scene complexity by reducing the amount of geometry to be processed by the graphics pipeline, while maintaining good visual fidelity. Such approaches include level-of-detail (LOD) control in rendering; see for example [1, 13]. In a recent work, [12] employs a multi-level pyramid representation, called geometry clipmaps, for the walkthrough of a large terrain with all its data compressed and loaded entirely in-core.

The query sub-problem involves the layout of data and the indexing of objects so that region queries can efficiently identify objects in the view frustum. Current approaches use spatial data structures such as quadtrees, octrees and R-trees that perform spatial partitioning and clustering of objects in the scene (see for example, [2, 3, 5]). These structures provides fast scene queries through their hierarchical properties to localize data currently in view (see [14]).

The pre-fetch sub-problem is the main focus of our paper. It concerns the retrieval of data from disk into core memory to fulfill future processing requirements. The objective is to ensure that at any one time, data required is already loaded in memory. Some existing out-of-core applications leave it to the operating system to do the actual fetching (paging) of data during runtime [10, 11, 15]. There are also approaches that use speculative pre-fetching depending on the viewer's current location and velocity to fetch data needed for future frames [4, 5, 7]. These approaches are sometimes coupled with sophisticated occlusion culling techniques; see [3, 5, 16]. However, it is not clear how they can provide specific quality-of-service guarantees with respect to the problem of page faults. Page faults, which happen when required data is not in the core memory, can adversely affect the frame rate required for a smooth walkthrough or flythrough.

The general lack of quantitative work addressing the pre-fetch sub-problem underlies our motivation to study it in detail. Our objective is to obtain good insights into pre-fetching mechanisms so as to build and optimize pre-fetching schemes. To this end, the paper first proposes formalism for the qualitative and quantitative analysis of the pre-fetch sub-problem, which takes into consideration the motion characteristics of the viewer, the data density of the virtual environment, and the data transfer capability of the disk. It then contributes in the analysis of two fundamental pre-fetching schemes, which are characterized by their uses of certain pre-assigned spatial or temporal thresholds. An interesting finding, as verified in our experiments with terrain walkthroughs, is the relationship between the data density supportable by pre-fetching scheme and the duration of time allocated to disk level pre-fetching.

The rest of the paper is organized as follows: Section 2 presents a survey of related previous work. Section 3 formulates the pre-fetch sub-problem. Section 4 discusses two basic pre-fetching schemes: spatial and temporal. Section 5 presents analysis for spatial and temporal threshold schemes, and Section 6 details experiments to verify the analysis. Section 7 concludes the paper.

2 RELATED WORK

Out-of-core visualization has received much research interest for over a decade. Early work on managing large amounts of data includes the work of Funkhouser *et al.* [9] on the walkthrough of large architectural models and Falby *et al.* [8] on a real-time 3D visual simulator. We shall focus on outlining approaches dealing with the pre-fetch sub-problem.

Recent works on handling large-scale datasets feature a general trend towards deploying advanced data layouts and index-

*{ngchumin, nguyenca, trandinh, tants}@comp.nus.edu.sg
†shinwe@gelement.com

ing schemes. Some of these methods rely on the data management capabilities of the operating systems to do demand paging (see [10, 15]). For example, the work of Lindstrom and Pascucci [11] has a new LOD scheme along with a data layout and indexing scheme using interleaved quadtrees. There is no explicit pre-fetching but the operating system handles the necessary paging of data. In the area of scientific visualization, Cox and Ellsworth [6] discuss the use of application specific knowledge to control the loading and unloading of data for better performance.

Works on pre-fetching schemes generally rely on multiple processes/threads, with one dedicated for pre-fetching. Varadhan and Manocha [16] present a system that uses a scene graph containing static LODs in a hierarchical LOD scheme. The system implements a priority-based pre-fetching algorithm which prioritizes objects in the scene based on some screen space error metric as well as their relative deviations from the viewer's line of sight. The pre-fetching routine then fetches multiple LODs for each object according to their priority in order to ensure that there is a high hit rate even when the object switches from one LOD to the next. The approach employs an expanded view frustum to reduce the page fault rate and the objects in the expanded view frustum are also prioritized according to their distance away from the actual view frustum. It pre-fetches data one frame in advance, and is implemented on an SGI workstation and an Onyx.

Unlike the above-mentioned pre-fetching methods that employ occlusion culling based on from-region visibility algorithms running on multiprocessor machines, Correa *et al.* [5] present a speculative pre-fetching system that works with from-point visibility algorithms and runs as a separate thread on a uniprocessor machine. The from-point pre-fetching method has several advantages, such as a shorter preprocessing time and better prediction on the set of objects to be pre-fetched. The system predicts the viewer's locations based on its current location as well as its translational and angular speeds.

Zach [17] describes a pre-fetching model using a conservative view frustum which is the union of all possible future view frustums within some time interval, based on the viewer's motion parameters. All data in the conservative view frustum, taking LOD selection and geomorphing into consideration, should be pre-fetched within a time interval of τ. Such τ should be as small as possible to improve path prediction, yet large enough to compensate for the high complexity of the selection process.

Our work aims to improve the understanding of pre-fetching mechanisms qualitatively and quantitatively, so as to design good pre-fetching schemes. This differs from all the above-mentioned works that focus on building pre-fetching systems for walkthrough and visualization; we analyze pre-fetching issues and design experiments to explore them. Unlike plain speculative pre-fetching, such as in the work of Chim *et al.* [4], our analysis also adapts the conservative view frustum approach in [17]. It differs from [17] in the analytical approach to derive a suitable time interval for pre-fetching to support high data densities.

3 Pre-Fetching Issues

The main objective of any pre-fetching scheme is to ensure that any required data at any time in the visual simulation is already resident in core memory. Failure of a pre-fetching scheme to meet the above objective results in the occurrence of *page faults*. Page faults cause either the visual simulation to be stalled waiting for data retrieval or computation involving those data to be skipped, resulting in degraded system performance. Therefore, the aim of pre-fetching schemes is to minimize the number of page faults with respect to a given operating environment (of viewer's motion) in a given system configuration (of available core memory and data transfer capability).

The execution flow of a pre-fetching scheme can be viewed as a chain of pre-fetches. Let S_i be the data in some region of the environment, called the *pre-fetch region*, to be fetched into core memory. It is determined at time t_i by the pre-fetching scheme while the viewer is at O_i, to support a certain amount of look-ahead in time. O_i is associated to S_i as its *reference point* and B_i its *boundary*. Figure 1 shows two successive pre-fetch regions S_i (solid oval) and S_j (dashed oval) with reference points O_i and O_j, respectively, and boundaries B_i and B_j, respectively.

In a usual pre-fetching scheme, all pre-fetching requests are strictly sequential where no new request can be initiated until an ongoing one has been completed. That is, the scheme has only 1 outstanding pre-fetch request at any time. Presenting this on a time line for any two successive pre-fetching regions S_i and S_j, we have $t_i < \bar{t}_i \leq t_j < \bar{t}_j$ where S_i and S_j are pre-fetched starting at t_i and t_j, and completely pre-fetched into core memory at $\bar{t}_i$ and $\bar{t}_j$ respectively.

Such an approach is obviously beneficial when multiple pre-fetching threads decrease the disk throughput. On the other hand, if the disk throughput can be improved due to the support of concurrent disk operations, we can execute our single pre-fetch operation over multiple threads to achieve better performance while treating it as one thread with higher data transfer capability in our following analysis.

With $t_i < \bar{t}_i \leq t_j < \bar{t}_j$, there are three time intervals of interest. During t_i to $\bar{t}_i$, only a part of the data in S_i is in the memory and the remaining is being fetched. During $\bar{t}_i$ to t_j, S_i is completely in the memory. From t_j to $\bar{t}_j$ more and more data in $S_j - S_i$ is progressively being fetched into the memory. However, in general, there may not be guarantee of these (partial) data being immediately available and useful to the application, hence a pre-fetching scheme should not depend on them and should regard data in $S_i \cap S_j$, which are in memory at t_j, as the only available data from t_j to $\bar{t}_j$.

3.1 Pre-fetch Mechanism

With the above notation, we proceed to analyze, in general, the working of a (deterministic) pre-fetch mechanism with the aim of minimizing the occurrences of page faults. Note that for a given pre-fetching scheme, variables such as S_i and t_i are dependent on the given walkthrough or flythrough ζ. In this section, we explicitly tag ζ onto the variables such as $S_i(\zeta)$, $t_i(\zeta)$ to denote this dependency.

The objective of pre-fetching can be stated as follows: during any walkthrough ζ, data in the view frustum at time t, denoted as $F_t(\zeta)$, is resident in memory, i.e,

$$\forall \zeta, \forall t, F_t(\zeta) \text{ is in core memory} \tag{1}$$

This is equivalent to the following statement which can be used to design pre-fetching mechanisms:

$$\forall \zeta, \forall t, t_i(\zeta) \leq t \leq \bar{t}_j(\zeta) \rightarrow F_t(\zeta) \subseteq S_i(\zeta) \tag{2}$$

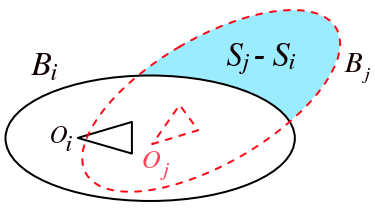

Figure 1: At time t_j, the pre-fetching scheme decides to fetch S_j. As some part of S_j is already in memory, pre-fetching is only needed for $S_j - S_i$. Note that the two triangles represent the view frustums at O_i and O_j, respectively.

See Appendix A for the proof of the above equivalence. The important implication of statement (2) is that it allows a pre-fetching scheme to work with discrete time intervals in practice. That is, from the viewers motion, a pre-fetching scheme can calculate, for any particular time t_i, the various possible positions of the viewing frustums within a period of time δt into the future, and thus determine S_i (and thus the particular shape of S_i) with a guarantee that no page fault will occur from t_i to $t_i + \delta t$. Clearly, the size of S_i must not exceed the portion of core memory designated to pre-fetch.

3.2 Disk Constraint

It is easy to realize that the performance of pre-fetching is dependent on the characteristics of the viewer's motion, the data distribution of the virtual environment, as well as the disk transfer rate. Formally, the relationship among the above factors can be stated with the following simple equation:

$$\sigma(S_j - S_i) \leq H(\tau) \tag{3}$$

where the *data transfer capability* $H(\tau)$ is defined to be the amount of data that can be transferred into core memory in time τ, and $\sigma(S_j - S_i)$ is the largest possible amount of data in $S_j - S_i$. This data is derived from the motion characteristics of user. To ensure a smooth walkthrough, equation (3) must be observed at all time. We note here that our analysis is independent of data layout since any choice of a particular layout scheme essentially translates to modifying the data transfer capability.

When a new pre-fetch is issued before an outstanding pre-fetch completes loading all requested data, we say there is a *scheme failure*. A scheme failure may not result in a subsequent page fault as those pages that have not been fetched may not be needed at all. Although scheme failures are tolerable in practice, any guarantee on system performance will be lost if they occur. On the other hand, a page fault is a result of a scheme failure. Hence, we restate the aim of a pre-fetching scheme as avoiding scheme failures, given an operating environment and system configuration.

4 Pre-fetching Schemes

In this section, we discuss two main classes of pre-fetching schemes: spatial threshold in Section 4.1 and temporal threshold in Section 4.2. A qualitative comparison between them is given in Section 4.3. The following are two reasonable assumptions which enable us to do a qualitative discussion here and a quantitative analysis in Section 5.

First, a virtual environment generally has varying densities over different regions. For a pre-fetching mechanism to avoid failures, it has to be capable of handling the region with highest density. Thus, in our analysis we assume that the virtual environment has a uniform data density ρ, defined to be the amount of data per unit area. In practice, ρ is set to be the highest density over the entire virtual environment. We say that a pre-fetching scheme can *support* a data density ρ if no scheme failure occurs when the scheme runs on a dataset of density ρ. In this case, ρ is *supportable* by the scheme.

Second, we consider schemes that maintain an invariant shape of pre-fetch region. That is to say that, for instance, the region covered by S_i overlaps exactly with the region covered by S_j under some translation and rotation.

4.1 Spatial Threshold Schemes

Following the discussion in Section 3, one way to categorize pre-fetching schemes is based on their decisions on (a) whether to trigger pre-fetching at current time t_j and (b) if so, the amount of data needed to be pre-fetched. For a pre-determined shape of the region covered by S_i (which is the most recent pre-fetching before time t_j), both decisions depend mainly on the distance of the current view frustum to the boundary B_i of S_i. The reason is as follows: the nearer the distance, the less amount of time available for pre-fetching before the view frustum moves out of B_i to result in a scheme failure, and the larger the amount of data in $S_j - S_i$ to be fetched.

As such, one can employ a closed boundary at some constant distance away from B_i to be a threshold b_i for a *general spatial threshold scheme*: the system does not perform data fetches until the current view frustum touches the threshold b_i. When it does so at time t_j, it defines a new reference point O_j at the viewer's location to calculate B_j, set a new threshold b_j, and start fetching $S_j - S_i$. Our choice of constant distance away from the boundary for the threshold is explained next. In the worst-case scenario, the viewer always moves towards the point of shortest distance from the threshold to the boundary. Thus, an arbitrary threshold is not better than a threshold of constant distance (equals to the shortest distance) from the boundary. Refer to Figure 2(a) for a possible sequence of events in a general spatial threshold scheme.

Let τ_b be the shortest possible amount of time the view frustum at O_i takes to touch the threshold b_i, and let τ be the shortest possible amount of time from the instance the view frustum touches b_i to the instance when it touches B_i, i.e. the duration of time allowed to complete fetching $S_j - S_i$. By statement (2), S_i (and, similarly, S_j) must contain sufficient data to support at least $\tau_b + \tau$ time starting at O_i (and, similarly, O_j as defined when the view frustum touches some point on b_i). For a given τ, by varying τ_b, we can obtain various sizes of pre-fetch regions. But, we must have $\tau_b \geq \tau$; otherwise, in the worst case, when the scheme uses up τ_b time to fetch a part but yet the whole of S_j, the view frustum can already touch b_j and thus trigger the scheme to request for a new pre-fetching while one is still on-going. This is not allowed as all such requests are strictly sequential. For a given τ, we define the spatial threshold scheme $\mathbb{S}$ as the general spatial threshold scheme where $\tau_b = \tau$.

4.2 Temporal Threshold Schemes

For the above spatial threshold scheme $\mathbb{S}$, the "busiest" situation is when it finishes a pre-fetch while the view frustum just touches the threshold of the pre-fetch region, thus initiating a new pre-fetch request immediately after the previous request finishes. In this case, $\mathbb{S}$ pre-fetches in every τ interval. Similar in spirit to the "busiest" situation of $\mathbb{S}$, a *temporal threshold scheme* $\mathbb{T}$ does the pre-fetching at some regular interval of τ. To initiate a pre-fetching at time t_j, it also sets the new reference point O_j at the current location of the viewer to calculate B_j so as to fetch $S_j - S_i$ while S_i has sufficient data to last till $t_j + \tau$ (and S_j will contain enough data to last till $t_j + 2\tau$). The scheme does not set any physical threshold, but is implemented with system interrupt at regular interval τ to trigger pre-fetching. See Figure 2(b) for a possible sequence of events of such a scheme.

4.3 Between Spatial and Temporal

For a given τ, the spatial threshold scheme $\mathbb{S}$ converges in its worst case (busiest situation) to the temporal threshold scheme $\mathbb{T}$, but $\mathbb{S}$ and $\mathbb{T}$ in general are different in that $\mathbb{S}$ needs to always fetch the largest region while $\mathbb{T}$ does not necessarily do so. On the other hand, $\mathbb{T}$ performs pre-fetching more frequently than $\mathbb{S}$. In some sense, (diligent) $\mathbb{T}$ does pre-fetching more frequently but fetches less each time, while (lazy) $\mathbb{S}$ only pre-fetches when needed but fetches more at one go. In practice, due to unforeseen circumstances, such as when the disk under-performs, $\mathbb{S}$ is more prone to causing scheme failures than $\mathbb{T}$.

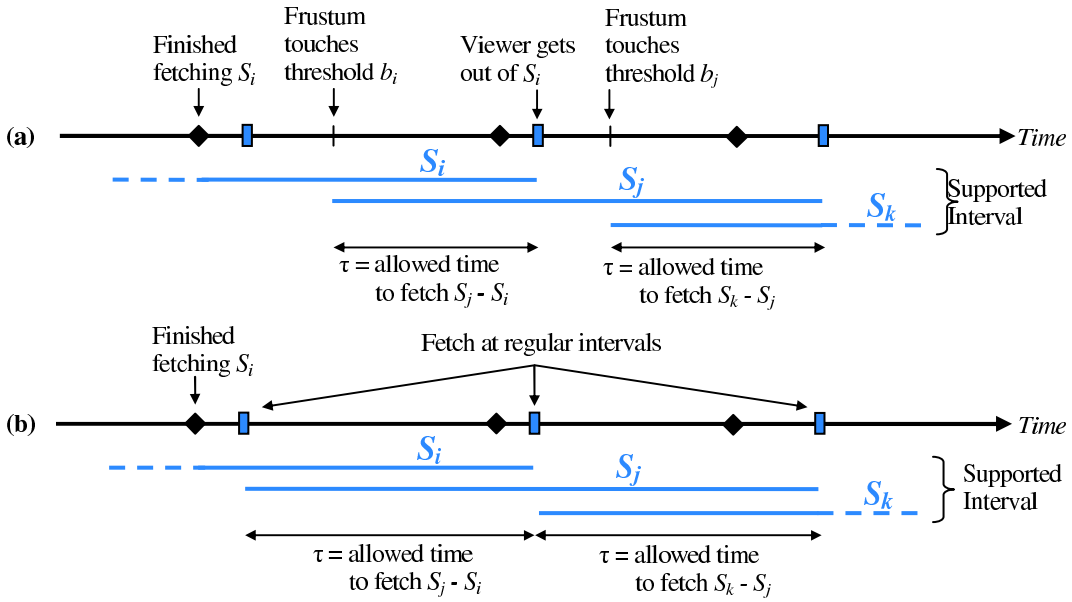

Figure 2: Possible sequences of events in (a) a general spatial and (b) a temporal threshold scheme.

5 The Pre-fetch Duration τ

This section derives a relationship between ρ, the data density supportable, and τ, the duration of pre-fetching time available in a general spatial threshold scheme. This derivation is done on a virtual environment that can be represented as a 2D map such as a terrain model, with the viewer's motion governed by a maximum translational speed v and a maximum angular speed ω. We note that the analysis is also applicable to the spatial threshold scheme $\mathbb{S}$ by choosing the appropriate $\tau_b = \tau$, and to the temporal threshold scheme $\mathbb{T}$ as $\mathbb{S}$ converges to $\mathbb{T}$ in its "busiest" situation.

5.1 Shapes of Pre-fetch Region

At time t_j, a pre-fetching scheme calculates S_j. We would like S_j to support a smooth walkthrough for some duration of time, say δt, into the future. By statement (2), it must at least contain the region that the view frustum can possibly cover within δt time starting from time t_j. With the constraints on the viewer's motion, interesting shapes of such region for our analysis are the fan shape (as shown in Figure 3(a), which is the most conservative region) and the circle shape (as shown in Figure 3(b), which extend the most conservative region of the fan shape to further cache all data needed in all possible rotational motions).

Fan Shape. Let O be the reference point of a pre-fetch region of the viewer. The current view frustum with field of view α is shown shaded as an isosceles triangle with two equal sides of length l and the third side representing the far plane. For simplicity, we ignore the small difference in the view frustum due to the truncation effect of the near plane. The whole region as shown is a fan shape, bounded by the arc $\overset{\frown}{DC}$ of radius $r = v\,\delta t$ with center O, $\overline{CB}$, $\overset{\frown}{BA}$ of radius $R = r + l$ with center O, and $\overline{AD}$. It defines the region needed by the viewer for a δt time interval where δt is the time needed for its view frustum to reach the boundary of the pre-fetch region. Specifically, the region is obtained by rotating the view frustum by the maximum amount $\beta = \omega\,\delta t$ to cover the space due to the rotational motion, and then translating this space in any direction away from O with the maximum amount r. We note that $\overline{OM}$ is parallel to $\overline{DA}$, so $\beta_1 = \beta + \sin^{-1}(r/R)$. The small shaded region at A's corner (and similar at B's) is not reachable within δt time but is included in the pre-fetch region for simplicity in our calculation.

Circle Shape. We extend the fan shape to a circle so as to cache all data needed in all possible rotational motions for any duration of time into the future while the position of the viewer remains unchanged; see Figure 3(b) where the circle shape is extended (as shown shaded) from the fan shape $ABCD$. (It is not possible to cache the same way for the translational motion unless the core memory is big enough to store the complete dataset.) As a result, it can reduce the amount of data to be pre-fetched per unit time (at the expense of using more core memory, as shown in Figure 5). When δt is large enough, the circle itself is the most conservative region to avoid scheme failures.

Our analysis does not consider the case where $\angle AOB > \pi$ in the fan shape (in Figure 3(a)). This is because such a shape is complicated to analyze and it does not have the advantages of a circle shape even though it caches more data than the fan shape. Thus, it is unlikely to result in better support of a high ρ.

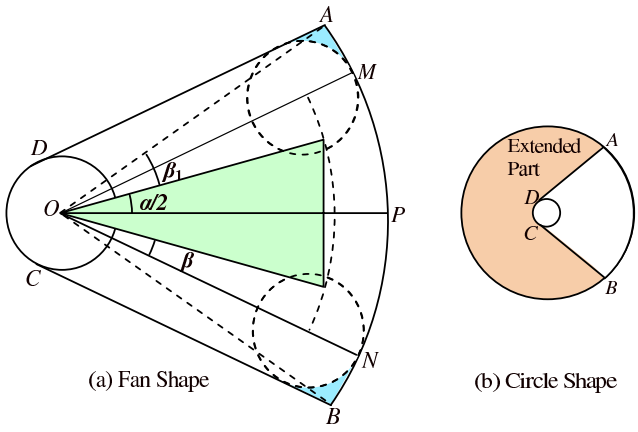

Figure 3: Shapes of pre-fetch regions.

5.2 $S_j - S_i$ of Fan Shape

At time t_i, the viewer is at $O_i = O$ and the system, which has a part of S_i in its core memory, does pre-fetching from t_i till $t_i + \tau$ to obtain the complete S_i. At time $t_j = t_i + \tau_b$, the viewer is assumed to have moved to $O_j = O'$, with its view frustum touching b_i in the worse case. At this point, the system has a full S_i (as $\tau_b \geq \tau$) but only a part of S_j in memory. Thus it needs to start pre-fetching from this time till time $t_i + \tau_b + \tau$ to obtain the complete S_j while the view frustum is still guaranteed to be inside S_i. In the following, we want to calculate the maximum region covered by $S_j - S_i$, denoted by T_{max}, for some given τ_b and τ, then derive a τ_b that minimizes T_{max} for each fixed τ.

Refer to Figure 4(a). The transformation from S_i to S_j can be separated into 2 parts: the rotation by angle $\gamma \leq \omega\tau_b$ about O and the translation along vector $\vec{p} = \overrightarrow{OO'}$ (with magnitude $p = |\vec{p}| \leq v\tau_b$). The rotation transforms B to B_1, A to A_1 and D to D_1. Then the translation transforms A_1 to A', B_1 to B', D_1 to D', and E_1 to E'. Let I be the intersection point of $\overline{A_1D_1}$ with the line segment extended from $\overline{AD}$ beyond D, and let λ be the angle $\angle O'Ox$. The amount of data to pre-fetch is $S_j - S_i$, as shown in five regions. To calculate the areas of the regions, we use the simple lemma as stated in Appendix B to get:

$$
\begin{aligned}
T_1 &= \text{area bounded by } \overset{\frown}{A'B'}, \overline{B'B_1}, \overset{\frown}{B_1A_1} \text{ and } \overline{A_1A'} \\
&= p|\overline{B_1A_1}|\sin(\angle(\vec{p}, \overrightarrow{B_1A_1})) \\
&= 2pR\sin(\beta_1 + \frac{1}{2}\alpha)\cos(\gamma - \lambda)\,. \\
T_2 &= \text{area of the parallelogram, } D_1D'A'A_1 \\
&= p|\overline{D_1A_1}|\sin(\angle(\vec{p}, \overrightarrow{D_1A_1})) \\
&= p\sqrt{R^2 - r^2}\sin(\lambda - (\frac{1}{2}\alpha + \beta + \gamma))\,. \\
T_3 &= \text{area bounded by } \overset{\frown}{E'D'}, \overline{D'D_1}, \overset{\frown}{D_1E_1} \text{ and } \overline{E_1E'} \\
&= p|\overline{D_1E_1}|\sin(\angle(\vec{p}, \overrightarrow{D_1E_1})) = pr(1 - \cos(\angle E_1OD_1)) \\
&= rp(1 - \cos(\frac{1}{2}\alpha + \beta + \gamma - \lambda))\,. \\
T_4 &= \text{area bounded by } \overline{IA_1}, \overset{\frown}{A_1A} \text{ and } \overline{AI} \\
&= \frac{1}{2}|\overline{IA_1}|\,|\overline{A_1A}|\sin(\angle(\overrightarrow{IA_1}, \overrightarrow{A_1A})) + \frac{1}{2}R^2(\gamma - \sin\gamma) \\
&= \frac{1}{2}\gamma R^2 - r^2\tan(\frac{1}{2}\gamma)\,. \\
T_5 &= \text{area bounded by } \overset{\frown}{DD_1}, \overline{D_1I} \text{ and } \overline{ID} \\
&= |\overline{ID}|\,|\overline{DO}| - \frac{1}{2}\gamma r^2 \\
&= r^2(\tan(\frac{1}{2}\gamma) - \frac{1}{2}\gamma)\,.
\end{aligned}
$$

To address the worst case, we calculate the maximum region covered by $S_j - S_i$. Notice that T_1, T_2, and T_3 are functions of $\vec{p}$, γ, and λ, while T_4 and T_5 are those of γ. So, to obtain T_{max}, we maximize $T_1 + T_2 + T_3$ and $T_4 + T_5$:

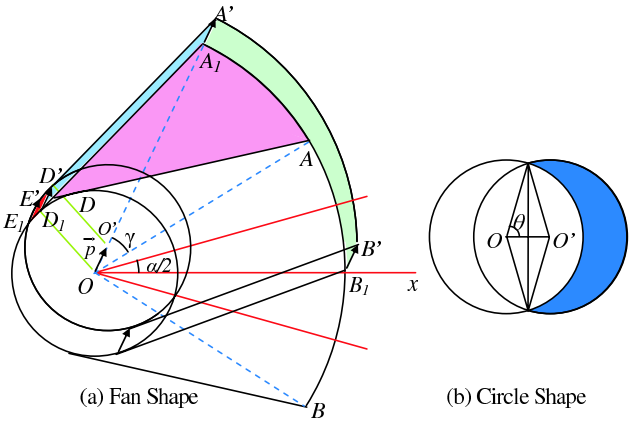

Figure 4: Regions (shaded) covered by $S_j - S_i$.

$$
\begin{aligned}
T_1 + T_2 + T_3 &= pr + pR\sin(\frac{\alpha}{2} + \beta + \sin^{-1}(\frac{r}{R}) - \gamma + \lambda) \\
&\leq pr + pR \leq v\tau_b(r + R)
\end{aligned}
$$

and $\quad T_4 + T_5 = \frac{1}{2}\gamma(R^2 - r^2) \leq \frac{1}{2}\omega\tau_b(R^2 - r^2)$.

So, $T_{max} = v\tau_b(r + R) + \frac{1}{2}\omega\tau_b(R^2 - r^2)$ when $\gamma = \omega\tau_b$, $p = v\tau_b$ and $\lambda = \frac{1}{2}\pi - (\frac{1}{2}\alpha + \beta + \sin^{-1}(\frac{r}{R}) - \omega\tau_b)$. If τ is fixed, T_{max} is an increasing function of τ_b for $\tau_b \geq \tau$ and is thus minimized when $\tau_b = \tau$. This is the case of the spatial threshold scheme $\mathbb{S}$. In this case, we have T_{max} as a function of τ :

$$
T_{max}(\tau) = (4v^2 + 2\omega vl)\tau^2 + (vl + \frac{\omega l^2}{2})\tau\,. \tag{4}
$$

5.3 $S_j - S_i$ of Circle Shape

Refer to Figure 4(b). The new region (shaded) $S_j - S_i$ swept out as the reference point changes from O to O' is via a translation along the vector $\vec{p} = \overrightarrow{OO'}$ (with magnitude $p = |\vec{p}| \leq v\tau_b$). Note that rotation has no contribution to $S_j - S_i$.

Then, the region covered by $S_j - S_i$ is:

$$
T = (\pi - 2\cos^{-1}(\frac{p}{2R}))R^2 + p\sqrt{R^2 - \frac{p^2}{4}}\,.
$$

It is easy to see that T achieves its maximum value when OO' is maximized; i.e. $p = v\tau_b$. So,

$$
T_{max} = (\pi - 2\cos^{-1}(\frac{v\tau_b}{2R}))R^2 + v\tau_b\sqrt{R^2 - (\frac{v\tau_b}{2})^2}.
$$

As in the case of the fan shape, T_{max} for the circle shape is an increasing function of τ_b for $\tau_b \geq \tau$ and thus is minimized when $\tau_b = \tau$. So, we have T_{max} as a function of τ:

$$
\begin{aligned}
T_{max}(\tau) = (\pi - 2\cos^{-1}(\frac{v\tau}{2(2v\tau + l)}))(2v\tau + l)^2 \\
+ v\tau\sqrt{(2v\tau + l)^2 - (\frac{v\tau}{2})^2}\,.
\end{aligned} \tag{5}
$$

From the above discussion, we conclude that among all the general spatial threshold schemes using fan shape or circle shape with fixed τ, the spatial threshold scheme $\mathbb{S}$ supports the highest density.

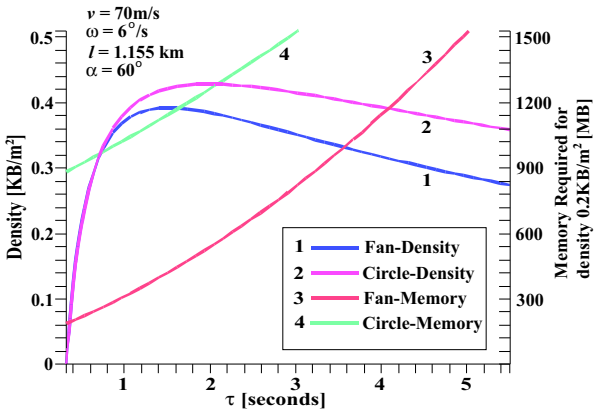

Figure 5: Data density supportable and the amount of core memory required by different pre-fetch duration τ.

5.4 Data Density Supportable

We first examine equation (4) to understand the highest data density supportable by a scheme with the fan shape. Suppose the disk data transfer capability H is the ideal case of $H(\tau) = K\tau$ for some constant K. In practice, there are additional delays due to various overheads, such as disk seek latencies, which is temporarily overlooked. Then from equations (3) and (4), we have:

$$\rho \leq \frac{K\tau}{T_{max}} = \frac{2K}{(8v^2+4\omega vl)\tau + (2vl+\omega l^2)} \leq \frac{K}{vl + \frac{\omega l^2}{2}}.$$

It is interesting to notice that vl is the area covered by the translational motion and $\frac{\omega l^2}{2}$ by the rotational motion per unit time, and ρ is no larger than the data transfer rate K over the sum of these areas.

Now suppose H is a more realistic function $H(\tau) = K(\tau - \varepsilon)$ where ε is the system overhead such as the seek time. Then,

$$\rho \leq \frac{K(\tau-\varepsilon)}{T_{max}} = \frac{2K(\tau-\varepsilon)}{(8v^2+4\omega vl)\tau^2 + (2vl+\omega l^2)\tau}. \quad (6)$$

The data density supportable is a function of τ, shown as a curve (labelled 1) in Figure 5 for a particular operating environment. Simple calculus shows that the highest data density supportable is at:

$$\tau_{max} = \varepsilon + \sqrt{\varepsilon^2 + \frac{2vl+\omega l^2}{8v^2+4\omega vl}\varepsilon}. \quad (7)$$

The function increases very rapidly to support larger data density when τ increases towards τ_{max}, and then it decreases gradually for τ moving away from τ_{max}. This is verified experimentally as discussed in the next section.

Similarly, we can obtain a curve (labelled 2) based on equation (5) for the circle shape as shown in Figure 5. The figure also shows that the circle shape can result in higher data density supportable than the fan shape does for the same set of parameters. This is the case when the rotational motion ω plays a more significant role compared to the translational motion v. Such situation may not be true when the translational motion v dominates region covered by $S_j - S_i$ as in our experiments (Section 6). Nevertheless, the circle shape requires much more core memory (curve 4) than that for the fan shape (curve 3) as shown in Figure 5.

5.5 Possible Extensions

The above calculation is a showcase of a methodology to analyze pre-fetching schemes. It can be extended with appropriate modifications to cater to specialized classes of viewer's motion such as in 3D maps or flythrough applications. In case of 3D maps, it is undoubtedly complex but remains possible to obtain a cubic function of τ for T_{max} and thus a relationship between data density ρ and τ.

Also, it is possible to analyze maps with a simple type of LOD support where progressively lower LOD is used with increasing distance from the viewer. In such a case, the calculation of $S_j - S_i$ for the highest LOD remains the same while the subsequent LODs are done in a simple way. T_{max} is the union of all data at all LODs. For example, in a 2D map, data within 200m is to be supported at maximum detail; data between 200m to 600m, at half detail; and data from 600m to 1km, at quarter detail. Then with the same set of motion parameters as in Figure 5, the analysis shows that the highest density supportable for the highest LOD is twice that of the case without LOD support.

6 Experiments on Terrain

Our experiments are conducted on two different dual-processor PC systems, an Intel Xeon 2.8GHz with 2GB (PC2100 266MHz SDRAM) of core memory on an Iwill DP533 motherboard running Windows XP with data stored in a 36GB Maxtor Atlas 10K IV U320 SCSI disk (10000 RPM, 4.3ms average seek time), and an Intel Xeon 2.1GHz with 2GB (PC800 400MHz RDRAM) of core memory on a Supermicro SUPER P4DC6 motherboard running Windows 2000 with a 80GB Seagate ST380021A IDE disk (7200 RPM, 9.5ms average seek time). We refer to the two systems in the experiment as IDE-2.1 and SCSI-2.8. The OS disk caching mechanisms for each system is set to on or off to get four different configurations in total.

Following our analysis in Section 5, our experiments are conducted in the context of a 2D terrain walkthrough. We generated terrain datasets of different data densities of up to a total of 20 GB on disk, stored as a grid of cells where each is a 64m×64m square region. Pre-fetching is done for data in cells overlapping the region covered by $S_j - S_i$. We use temporal pre-fetching scheme with a fan shape; it is implemented as a separate thread from the rendering thread. A fixed pool of core memory is used for the management of terrain data. To realize the worst case scenario, we program the viewer's motion to follow a path that causes the maximum amount of data to be transferred on each pre-fetch request. There are in total 100 pre-fetches in our chosen path. The circle shape is not experimented as it would require much larger amount of core memory to create scheme failures.

The role of τ. For each configuration of our systems, we collected statistics on its number of scheme failures for terrains of different data densities. Two sets (out of four) of data, one with disk caching mechanism enabled and one without, are shown in Figure 6; the two other sets of data are not shown here, but reflect similar trends. In each chart, we show four representative data density curves out of the many different ones that we have collected. These statistics testify the theoretical prediction in the previous section that for each data density, there is a range of τ that achieves close to zero scheme failures. Another finding is that the higher the data density, the smaller the range of such good τ.

Highest Data Density ρ. For each τ, we ran experiments for a range of data densities to determine the highest ρ for which no scheme failure occurs (with allowance of occasional 1 scheme failure due to noise). Such values are used to plot Figure 7. The shapes of these curves conform to the theoretical one as shown in Figure 5.

Best range of τ. In addition, Figure 7 also shows that the good τ which support the highest density for IDE-2.1 (OS disk caching

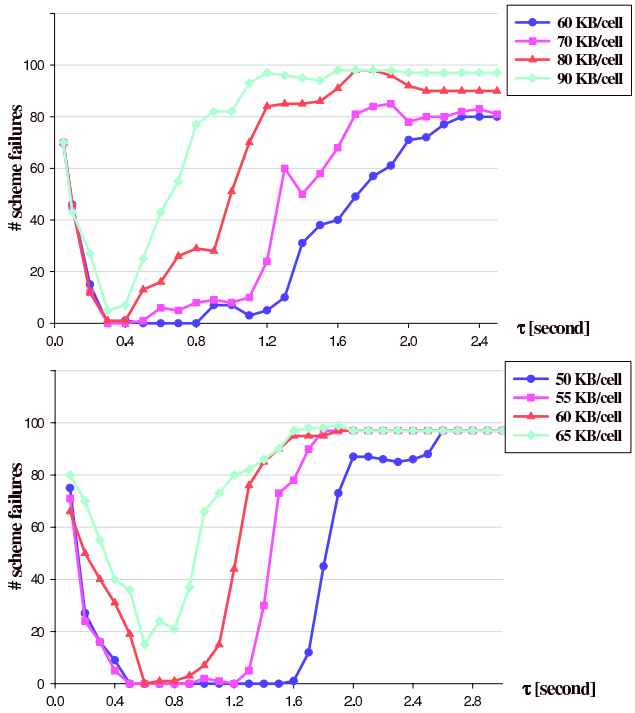

Figure 6: Scheme failures against τ with OS disk caching enabled(top)/disabled(bottom) for SCSI-2.8/IDE-2.1

disabled) is in the range of 0.5 to 0.8 second. For such a case where OS disk caching is disabled, we can derive a reasonable bound of ε between 0.22 to 0.31 through a linear regression on the hard disk performance statistics, and apply it to equation (7) to obtain the theoretical good τ of between 0.6 to 0.9 second. These two ranges of good τ in the case of SCSI-2.8 (with ε between 0.31 to 0.54 again obtained through linear regression) are from 0.6 to 0.9 second in the experiments and from 0.8 to 1.2 second in equation (7). When OS disk caching is enabled, we do not have a good bound on ε due to the large variance in disk performance, and thus cannot compare those values of τ in experiments to those in theory.

System Configuration. Referring to Figure 7, we note that OS disk caching helps to increase the highest density supportable for each τ. When the disk caching is disabled, the highest densities supportable on each configuration conform to those predicted by equation (6). In particular, for IDE-2.1, equation (6) gives between 65 to 70 KB/cell, while the experimental result is 60 KB/cell; for SCSI-2.8, between 70 to 80 KB/cell while the experimental result 62.5 KB/cell. Some discrepancy between the result of equation (6) and the experiments (besides unavoidable noise in the experiments) can be attributed to the fact that experimental data is grid based while the analysis assumes an infinitely fine grid.

7 Concluding Remarks

Overall, this paper presents both analytical and experimental studies on pre-fetching schemes. In particular, it formulates the pre-fetch problem, discusses design parameters of spatial and temporal pre-fetching schemes and relates data densities supportable by these schemes to the duration of time allocated to do pre-fetching. Our work serves to supplement the meager pool of knowledge in understanding pre-fetching quantitatively. The analysis also serves to showcase a methodology for understanding pre-fetching schemes. One implication of our work is that for each data set there is a need

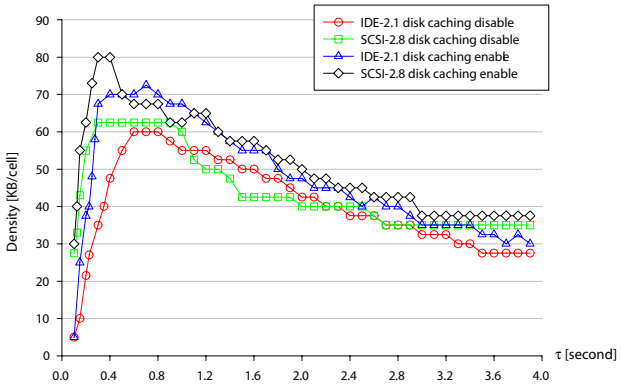

Figure 7: Highest density supportable for each τ.

to find the corresponding pre-fetch interval to achieve good performance.

With the insights provided by the theoretical analysis, we conclude by presenting some practical considerations to augment our pre-fetch mechanisms. For example, while our analysis assumes constant data density, a system can incorporate "lazy" pre-fetching which defers fetching requests in low density regions. This can potentially improve the performance of the system by delaying data transfers until they are absolutely needed. Alternatively, when lower densities are encountered, more data can be loaded in advance in anticipation of later use. Better performance can also be achieved by incorporating some form of path prediction to cull regions that are unlikely to be reached by the user. Other useful optimizations such as proximity-based prioritized fetching can also be incorporated to build a practical pre-fetching system. All these will potentially improve the performance of pre-fetching systems.

Acknowledgements

This research is supported by the National University of Singapore under grant R-252-000-181-112.

References

[1] Daniel Aliaga, Jon Cohen, Andrew Wilson, Eric Baker, Hansong Zhang, Carl Erikson, Kenny Hoff, Tom Hudson, Wolfgang Stuerzlinger, Rui Bastos, Mary Whitton, Fred Brooks, and Dinesh Manocha. MMR : An interactive massive model rendering system using geometric and image-based acceleration. In *Proceedings of the 1999 Symposium on Interactive 3D graphics*, pages 199–206. ACM Press, 1999.

[2] Xiaohong Bao and Renato Pajarola. LOD-based clustering techniques for optimizing large-scale terrain storage and visualization. In *Proceedings of the SPIE Conference on Visualization and Data Analysis*, pages 225–235, 2003.

[3] William V. Baxter, III, Avneesh Sud, Naga K. Govindaraju, and Dinesh Manocha. Gigawalk: Interactive walkthrough of complex environments. In *Proceedings of the 13th Eurographics Workshop on Rendering*, pages 203–214. Eurographics Association, 2002.

[4] Jimmy H. P. Chim, Mark Green, Rynson W. H. Lau, Hong Va Leong, and Antonio Si. On caching and prefetching of virtual objects in distributed virtual environments. In *Proceedings of the 6th ACM International Conference on Multimedia*, pages 171–180, 1998.

[5] W. T. Correa, J. T. Klosowski, and C. T. Silva. Visibility-based prefetching for interactive out-of-core rendering. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 1–8, 2003.

[6] Michael Cox and David Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proceedings of the 8th Conference on Visualization '97*, pages 235–244, 1997.

[7] Douglass Davis, William Ribarsky, Nickolas Faust, and T. Y. Jiang. Intent, perception, and out-of-core visualization applied to terrain. In *Proceedings of the Conference on Visualization '98*, pages 455–458, 1998.

[8] John S. Falby, Michael J. Zyda, David R. Pratt, and Randy L. Mackey. NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation. In *Computer & Graphics*, pages 65–69, 1993.

[9] Thomas A. Funkhouser, Carlo H. Sequin, and Seth J. Teller. Management of large amounts of data in interactive building walkthroughs. In *Proceedings of the 1992 Symposium on Interactive 3D graphics*, pages 11–20, 1992.

[10] Benjamin Gregorski, Mark Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings of the Conference on Visualization '02*, pages 475–484, 2002.

[11] Peter Lindstrom and Valerio Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. In *IEEE Transactions on Visualization and Computer Graphics*, pages 239–254, 2002.

[12] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics*, 23(3):769–776, 2004.

[13] David Luebke, Martin Reddy, Jonathan Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of Detail for 3D Graphics*. Elsevier Science, 2003.

[14] Renato Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In *Proceedings of the Conference on Visualization '98*, pages 19–26. IEEE Computer Society Press, 1998.

[15] Valerio Pascucci and Randall J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, 2001.

[16] Gokul Varadhan and Dinesh Manocha. Out-of-core rendering of massive geometric environments. In *Proceedings of the Conference on Visualization '02*, pages 69–76, 2002.

[17] Christopher Zach. Integration of geomorphing into level of detail management for realtime rendering. In *Proceedings of the 18th Spring Conference on Computer Graphics*, pages 115–122, 2002.

APPENDIX A

We provide the proof of the equivalence of statements (1) and (2).

Assume statement (1) holds but statement (2) does not for a given ζ and t. Then there is some $t_i(\zeta) \le t \le \bar{t}_j(\zeta)$ such that $F_t(\zeta) \not\subseteq S_i(\zeta)$. We first argue that t is not from $\bar{t}_i(\zeta)$ till $\bar{t}_j$. From $\bar{t}_i(\zeta)$ till t_j, the core memory contains only $S_i(\zeta)$, and from t_j till $\bar{t}_j$, the core memory contains some part (if not all) of $S_i(\zeta)$. And, $F_t(\zeta)$ is in the core memory as given by statement (1), thus $F_t(\zeta) \subseteq S_i(\zeta)$. As such, the mentioned t is such that $t_i(\zeta) \le t \le \bar{t}_i(\zeta)$. Now we consider a walkthrough or flythrough ζ' such that ζ and ζ' are identical until t but then the viewer remains stationary until time $\bar{t}_i(\zeta')$. Since ζ' and ζ are identical until t, $S_i(\zeta') = S_i(\zeta)$. Thus $F_{\bar{t}_i}(\zeta') = F_t(\zeta') = F_t(\zeta) \not\subseteq S_i(\zeta) = S_i(\zeta')$. This is a contradiction as at time $\bar{t}_i$, we have as we argued in the above that $F_{\bar{t}_i}(\zeta') \subseteq S_i(\zeta')$. Thus if statement (1) holds then statement (2) holds.

Conversely, assume that statement (2) holds. Then for all t, there are i and j such that: $t_i(\zeta) \le t \le \bar{t}_j(\zeta)$. There are three cases of time interval to consider. First, if $\bar{t}_i(\zeta) \le t \le t_j(\zeta)$, then $F_t(\zeta) \subseteq S_i(\zeta)$ as given and $S_i(\zeta)$ is in the core memory, thus $F_t(\zeta)$ is in the core memory. Second, if $t_j(\zeta) \le t \le \bar{t}_j(\zeta)$, then as this interval is a sub-interval of both $t_i(\zeta)$ till $\bar{t}_j(\zeta)$ and $t_j(\zeta)$ till $\bar{t}_k(\zeta)$ where t_k is the time where the next successive pre-fetching after S_j is performed, we have $F_t(\zeta)$ is a subset of both $S_i(\zeta)$ and $S_j(\zeta)$. Thus, $F_t(\zeta) \subseteq S_i(\zeta) \cap S_j(\zeta)$ and is in the core memory (even though the parts of $S_i(\zeta)$ that are not in $S_j(\zeta)$ have already been moved out of the memory). Third, if $t_i(\zeta) \le t \le \bar{t}_i(\zeta)$, this is just the same as the previous case by shifting the discussion to one preceding pre-fetching. This finishes the proof of the claim.

APPENDIX B

Lemma. *Suppose we have an arc $\widehat{AB}$, and its displacement by a vector $\vec{p} = \overrightarrow{OO'}$ results in the arc $\widehat{A'B'}$. If these two arcs do not intersect, then the area bounded by them and $\overline{AA'}$, $\overline{BB'}$ is $|\vec{p}| \times |\overrightarrow{AB}| \times \sin(\angle(\vec{p}, \overrightarrow{AB}))$.* □

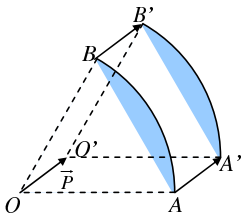

Figure 8: Illustration of lemma.