Prefetching in Visual Simulation

Chu-Ming Ng[+], Cam-Thach Nguyen[+], Dinh-Nguyen Tran[+], Shin-We Yeow*, Tiow-Seng Tan[+]

[+]National University of Singapore *G Element Pte Ltd

1 Motivation

This project examines the problem of visual simulation of virtual environment that is too large to fit into the main memory of a PC. We broadly classify the problem into three subproblems: render, query and prefetch, which correspond, respectively, to processing data to be displayed, identifying and organizing data to be retrieved, and retrieving (identified) data into main memory in anticipation of the need to render them in the near future. Unlike the first two subproblems, there is little existing work that reports the prefetch subproblem in detail. Some existing applications adopt advanced data indexing and layout in an application specific way but leave the operating system to do the actual fetching (paging) of data during runtime (see, for example, [LiPa02]). There are also approaches that use speculative prefetching based on the viewer's current position and velocity to prefetch data needed for future frames (see, for example, [CGLL98]). These are sometimes coupled with sophisticated occlusion culling techniques that reduce the amount of geometry that needs to be fetched from disk (see, for example, [VaMa02]). On the whole, the focus of current approaches is in solving the render and query subproblems but it is not clear how these methods can provide specific quality-of-service guarantee with respect to page fault rates. The general lack of quantitative work on the prefetch subproblem underlies our motivation to study it in detail.

2 Prefetching Issues

The main objective of any prefetching mechanism is to ensure that any data that are needed for processing during any time of the visual simulation are already loaded into the memory. Failure of the prefetching mechanism to maintain the above objective results in the occurrence of *page faults*. The aim of all prefetching mechanism is thus to minimize the number of page faults to support a given operating environment with a given system configuration. At any time t_i for the observer O, let S_i be the amount of data in the main memory M, and F the needed set of data in its current viewing frustum. Then, prefetching wishes:

$$F \subseteq S_i. \quad (1)$$

Also, $$S_i \subseteq M. \quad (2)$$

To maintain equation (1), one must perform some prefetching starting at some later time t to obtain S_j by time t_j. While prefetching is on-going, O continues with its movement. Then, S_i must be large enough to fulfill equation (1) till S_j is available, i.e.

$$F \subseteq S_i \text{ for all time till time } t_j. \quad (3)$$

and those data to be fetched (i.e. those in S_j but not in S_i) must not be larger than the amount of data that can be fetched from disk to the main memory from t till t_j:

$$S_j - S_i \le H(t_j - t) \quad (4)$$

Emails: {ngchumin | nguyenca | trandinh}@comp.nus.edu.sg, shinwe@gelement.com, tants@comp.nus.edu.sg,

where H is the system data transfer rate (see Figure 1). When equation (4) is not achieved by a prefetching request, we call it a *scheme failure*. A scheme failure at time t_j may not result in a page fault at time t_j as those pages that are yet to be fetched may not be needed yet. Though scheme failures may be tolerable, they result in no guarantee in system performance and thus should be avoided. On the other hand, a page fault is a result of a scheme failure. As such, we re-state the aim of prefetching mechanism as minimizing the number of scheme failures.

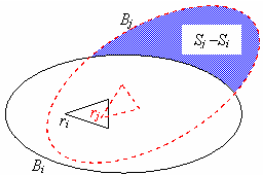

Figure 1. At time t, the prefetching mechanism decides to start prefetching to obtain S_j. As part of S_j is the same as that of S_i, the fetching only needed for the part in $S_j - S_i$.

3 Prefetching Schemes

For purposes of analysing prefetching schemes, we make the following assumptions. First, we assume it is a 2D map with uniform data density ρ. Though this is unlikely the case in practice, one choice is to set the density to be the highest density of the map. This is reasonable as in the worst case, O can spend all the time moving in the highest density part of the map. Second, we assume that a prefetching scheme maintains the same size of the S_j each time it calculates S_j. Third, at any time, there is at most one outstanding prefetching work. That is, no prefetching thread can be initiated until an ongoing prefetching has completed. If not, the analysis can be modified to as if there is only one pending prefetching work.

Shapes of Prefetch Region. We consider two shapes of prefetch region. First, we have the fan shape as shown in Figure 2. Suppose the motion of O is governed by its maximum speed v and view direction can change by a maximum angular speed ω. Then, the calculation of S_j at location r_j (see also Figure 1) is such that the shortest amount of time τ for the frustum of O to touch B_j starting at time t is the same in all directions for the given v and ω. Such τ is the amount of time that the system has enough data to run till time $t + \tau$ without fetching more data. Second, we have the circle shape as shown in Figure 3 where it extends the fan shape with extra data to make it a circle. The reason to consider the circle shape is that it eliminates the contribution of rotation to $S_j - S_i$, resulting in smaller amount of data need to be fetched each time. It, however, requires much more memory to work.

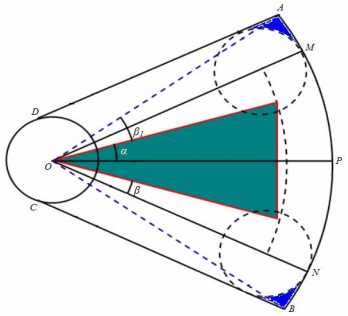

Figure 2. Fan shape where the center shaded triangle is the current viewing frustum of O.

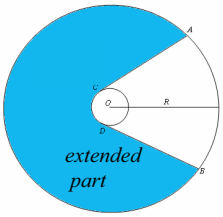

Figure 3. Circle shape extended from the fan shape.

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

One way to categorize prefetching schemes is to examine their decisions on (a) whether to triggle prefetching at current time t and (b) if so, the amount of data to be fetched, while honoring equations (1) to (4). For a pre-determined shape of S_i, both decisions depend on the only factor of the distance of the current frustum F to the boundary B_i of S_i. The reasoning being as follows: the nearer the distance, the less amount of time available for prefetching before O moves out of B_i to possibly result in a page fault, and the larger the amount of data in $S_j - S_i$ to fetch; see again Figure 1 for the illustration. We analyse the following two prefetching schemes.

Spatial Prefetching Scheme. The spatial prefetching scheme employs a closed curve to be a threshold boundary b_i as follows: The system does not perform data fetches until the current frustum touches the threshold boundary b_i. When it does at time t, it defines a new reference point r_j at the observer location to calculate B_j so as to fetch $S_j - S_i$ and to set the new threshold boundary b_j. It can be argued that the best spatial prefetching scheme $\mathbb{S}$ to support the largest ρ is one with the shape S where its reference point to the threshold boundary, and the threshold boundary to its boundary are both of time τ away, for a total of 2τ of data contained in S.

Temporal Prefetching Scheme. For the above spatial prefetching scheme $\mathbb{S}$, the "busiest" situation is when it finishes a prefetch, the frustum again touches the threshold boundary and thus immediately initiates a new prefetch, and so on. In this case, $\mathbb{S}$ prefetches in every τ interval. Same in spirit to the "busiest" situation of $\mathbb{S}$, a temporal prefetching scheme $\mathbb{T}$ does its prefetching at some regular interval of τ. To initiate a prefetch at time t, it also sets the new reference point r_j at the current location of the observer to calculate B_j so as to fetch $S_j - S_i$ where S_i has sufficient data to enable computation till $t + \tau$, and S_j will contain enough data to enable computation from $t + \tau$ till $t + 2\tau$. The scheme does not set any threshold boundary, but is to be implemented with system interrupt at regular intervals of τ to triggle prefetching.

4 Relationship between τ and ρ

This section presents a relationship between data density ρ and the amount of prefetching time τ available for the temporal prefetching scheme $\mathbb{T}$. The result also applies to the spatial prefetching scheme $\mathbb{S}$ as we discussed in the last section that $\mathbb{S}$ converges to $\mathbb{T}$ in the worst case. To obtain the mentioned relationship, we first study the maximum complement $S_j - S_i$ as in Figure 1 using simple geometry. Let l denote the distance of the far plane from the observer. We have:

(a) For fan shape:

$$\frac{S_j - S_i}{\rho} = (4v^2 + 2\omega v l)\tau^2 + (vl + \frac{\omega l^2}{2})\tau$$

(b) For circle shape:

$$\frac{S_j - S_i}{\rho} = (\pi - 2\cos^{-1}(\frac{v\tau}{2(2v\tau + l)}))(2v\tau + l)^2 + v\tau\sqrt{(2v\tau + l)^2 - (\frac{v\tau}{2})^2}$$

In the ideal case where the disk performance can be approximated as a linear function $H(\tau) = K(\tau - \varepsilon)$, where K and ε are constants, we can substitute the above into equation (4) to plot Figure 4 as shown. Because $(S_j - S_i)/\rho$ is a quaratic function of τ, while $H(\tau)$ is a linear function, a large τ may result in bad performance (small ρ supported). On other hand, if τ is too small, harddisk overhead contributes a big percentage in transfer time and result in bad performance. In other words, there is a range of suitable τ to be used to obtain good performance. This is conformed to the experiments discussed in the next section.

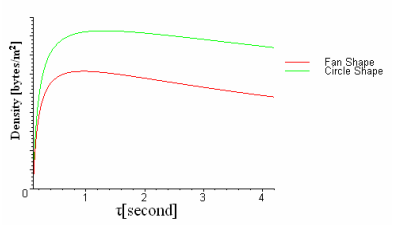

Figure 4. Density ρ can be supported as a function of τ.

5 Experiment on Terrain Walkthrough

In our experiments, terrain data are used as it is easy to create terrain datasets of different densities in a 2D map. Data are stored in a grid of cells. We use temporal prefetching scheme that is implemented as a thread separated from other threads such as the rendering one. To realise the worst case situation, we force the observer to run on a "tricky path" where the amount of data to be prefetched is maximum each time a prefetching is performed. We have run experiments on four densities, ranging from 75 to 112.5 Kbytes per cell with the chosen parameters indicated in the graph. Our preliminary experiment results conform to the theoretical prediction outlined in the previous section; that is, there is a good range of τ with small number of scheme failures.

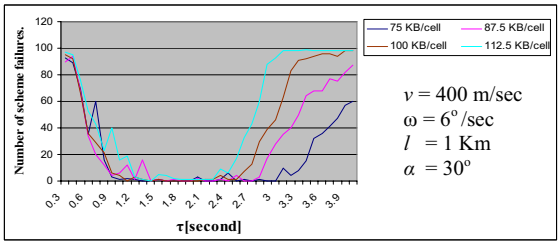

Figure 5. Experimental results on the number of scheme failures against τ.

6 Concluding Remarks

Our work aims to supplement the meager pool of knowledge in understanding prefetching quantitatively. With this, one can incorporate other practical consideration on building prefetching systems to meet other challenges in real applications. We intend to do further experimentation in different platforms. Also, there are a lot of further works. One possible direction is to incorporate the understanding into building a practical prefetching system as mentioned in the above. Such practical system may support certain path predictions, "urgent" queue prefetching for those page faults, selective memory release when prefetching new data, special data organization that incoporate LOD and occlusion [BaPa03] etc.

References

[BaPa03] X. Bao and R. Pajarola. "LOD-based Clustering Techniques for Optimizing Large-scale Terrain Storage and Visualization", *Proc. SPIE Conference on Visualization and Data Analysis*, 2003.

[CGLL98] H. Chim, M. Green, W. Lau, H. Leong and A. Si. "On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments", *Proc. ACM Multimedia*, pp. 171—180, 1998.

[LiPa02] P. Lindstrom and V. Pascucci. "Terrain Simplification Simplified: A General Framework for View Dependent Out-of-Core Visualization", *IEEE Transactions on Visualization and Computer Graphics*, 8(3), July-September 2002, pp. 239—254.

[VaMa02] G. Varadhan and D. Manocha. "Out of Core Rendering of Massive Geometric Environments", *Proc. IEEE Visualization*, pp. 69—76, 2002.