Eurographics/ IEEE-VGTC Symposium on Visualization (2006)
Thomas Ertl, Ken Joy, and Beatriz Santos (Editors)

Knowledge-Based Out-of-Core Algorithms for Data Management in Visualization

David Chisnall[1], Min Chen[1] and Charles Hansen[2] †

[1]Department of Computer Science, University of Wales Swansea, UK
[2]School of Computing, University of Utah, USA

Abstract

Data management is the very first issue in handling very large datasets. Many existing out-of-core algorithms used in visualization are closely coupled with application-specific logic. This paper presents two knowledge-based out-of-core prefetching algorithms that do not use hard-coded rendering-related logic. They acquire the knowledge of the access history and patterns dynamically, and adapt their prefetching strategies accordingly. We have compared the algorithms with a demand-based algorithm, as well as a more domain-specific out-of-core algorithm. We carried out our evaluation in conjunction with an example application where rendering multiple point sets in a volume scene graph put a great strain on the rendering algorithm in terms of memory management. Our results have shown that the knowledge-based approach offers a better cache-hit to disk-access trade-off. This work demonstrates that it is possible to build an out-of-core prefetching algorithm without depending on rendering-related application-specific logic. The knowledge based approach has the advantage of being generic, efficient, flexible and self-adaptive.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques – Graphics data structures and data types; I.3.m [Computer Graphics]: Visualization – Point-based techniques; D.4.2 [Operating Systems]: Storage Management – Allocation/deallocation strategies.

1. Introduction

Never before in history have we had such capability for generating, collecting and storing digital data. Data repositories at terabyte level are becoming commonplace in many applications, including bioinformatics, medicine, remote sensing and nano-technology. In some applications, such as network traffic visualization [Kou99] and video visualization [DC03], we are encountering the scenario that dynamic data streams are almost temporally unbounded.

Data management is the very first issue in handling very large datasets. Many visualization processes involve datasets that are much too large for the internal memory of a computer, and have to rely on external disk storage, usually under the virtual memory management of an operating system. The external disk access can become a serious bottleneck in terms of rendering speed. *Out-of-core algorithms* (also known as *external memory algorithms*) [Vit01] are designed to solve a variety of batch and interactive computational problems by minimizing disk I/O overhead. Many problem-specific out-of-core algorithms (e.g., [CMPS96, SCM99, FS01]) were proposed, showing noticeable advantages over direct reliance on operating systems.

Such advantages are not in any way unexpected since the virtual memory management algorithm in an operating system is not coded with any *application-specific logic*, such as a data partitioning structure or a rendering algorithm. In most modern operating systems, the memory management algorithms are designed to be generic, efficient and often self-adaptive. Many have incorporated prefetching strategies for anticipatory memory management. Hence this raises an interesting question: *If a relatively generic algorithm is able to acquire some application-specific logic dynamically, would it be able to provide support to the application con-*

† Email: {csdavec,m.chen}@swansea.ac.uk, hansen@cs.utah.edu

cerned in a similar way to those algorithms with hard-coded application-specific logic?

The main objective of this work was to seek an answer to this question in the context of visualizing very large point datasets. In particular, we focused on algorithms that were capable of acquiring knowledge of the data access patterns dynamically and adapt their prefetching strategies accordingly. We considered four out-of-core algorithms:

- *The Least Recently Used (LRU) strategy* — This is a simple memory management algorithm discussed extensively in many textbooks on operating systems. This is the most generic among the four considered and contains a very limited amount of application-specific logic. It is used in this work as a base-line for evaluating all algorithms.
- *The Ray Driven Predictor (RDP)* — This is the least generic among the four considered. It relies on a significant amount of hard-coded application-specific logic, including both the data structures and the rendering algorithm, to make predictions.
- *The Access Path Predictor (APP)* — This algorithm assumes that the most likely access pattern is a predecessor-current-successor pattern. Unlike RDP, it does not attempt to hard-code such logic mathematically, and instead makes predictions based on previous access patterns. Hence, it is a knowledge-based algorithm.
- *The History-based Access Predictor (HAP)* — This algorithm also adopts a knowledge-based approach, but is more generic than APP without the assumption about the likelihood of any access pattern. It maintains a relatively fuller record of access history.

In order to facilitate the evaluation of these four algorithms, we considered the problem of rendering multiple point datasets in a volume scene graph as an example application. Through this example, we found that it is possible to achieve good performance in visualization using a data management algorithm which does not exploit the rendering-related logic. The data-structure-related logic, on the other hand, plays a more critical role in out-of-core strategies.

2. Related Work

The problem of insufficient memory has been around for almost as long as stored-program computers. Most operating systems provide some form of virtual memory [Den70] to help alleviate this problem. Unfortunately, visualization tasks, where the *working set* [Den68] changes rapidly, do not mix well with the the *demand paging strategy* [SGG00] used by most operating systems.

Cox and Elleworth [CE97] proposed a method by which the application could control the demand paging strategy, allowing data to be evicted in an intelligent way. The authors discovered in their analysis of demand paging algorithms in existing operating systems that a significant performance increase could be gained by using smaller page sizes.

An effective out-of-core [SCESL02], or external memory [Vit01] strategy requires an efficient prefetching algorithm (such as in [VM02]) in order to prevent disk latency being the limiting factor in rendering. Various out-of-core algorithms have been proposed in the context of visualization. However, they are mostly tailored to a specific rendering method. Methods have been proposed for both structured and unstructured 3D datasets, including: (i) isosurface extraction [CMPS96, CS97, CS98, CFSW03, SH00], (ii) terrain rendering [LP02], (iii) streamline visualization [USM97], (iv) mesh simplification [Lin00], (v) rendering time-varying volume data [SCM99], (vi) rendering unstructured volumetric grids [LM99, FS01, CFSW03], (vii) ray tracing [PKGH97], and (viii) radiosity [TFFH94].

While some algorithms rely little on internal memory (e.g., [CS97, FS01]), others utilize preprocessed data structures, such as octree [USM97] and indexing [SCM99] to optimize disk I/O operations.

3. Motivation and Example Application

The main focus of this paper is the design and evaluation of knowledge-based out-of-core algorithms. In particular, we would like to adapt the generic design principle for memory and cache management in modern operating systems, and reduce the amount of hard-coded application-specific logic in out-of-core algorithms for visualization systems. Typically application-specific logic falls into two categories, namely *data-related* and *traversal-related*. The former is concerned with the organization of data, such as data partitioning, connectivity and block size. The latter is concerned with the access sequence of different parts of the data.

In visualization, the most common traversal-related logic is *rendering-related*, and the majority of access is read-only, hence the so-called 'dirty data block' issue does not normally arise. If an out-of-core algorithm does not have rendering-related logic hard-coded, it can be applied to different visualization applications that have the same data-related logic, provided that it can acquire rendering-related logic dynamically at run-time, and it can adapt itself dynamically to varying parameters such as scene complexity, data size, viewing positions, sampling intervals and transfer functions. Meanwhile, when an out-of-core algorithm is aware of some data-related logic, it will most likely be more cost-effective in handling in-core and out-core swapping than the paging strategy built into an operating system.

In order to assist in the design and evaluation of the algorithm concerned, as an example application, we considered the problem of rendering multiple point datasets in a volume scene graph using discrete ray tracing, which was initially discussed in [Che05]. Although point datasets are commonly rendered using projective methods such as surfels [PZvBG00] and QSplat [RL00], There are many merits for deploying discrete ray tracing in conjunction with vol-

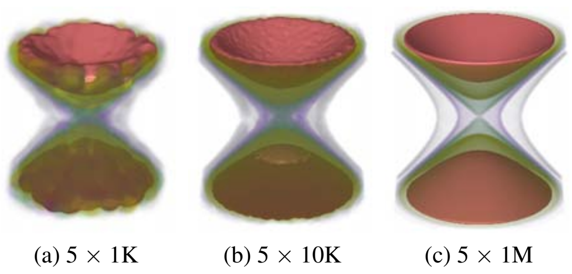

Figure 1: *Five iso-surfaces of a hyperbolic field are visualized using point sets that are randomly placed on the surfaces with different resolutions. Each iso-surface is approximated by (a) 1K, (b) 10K and (c) 1M points respectively.*

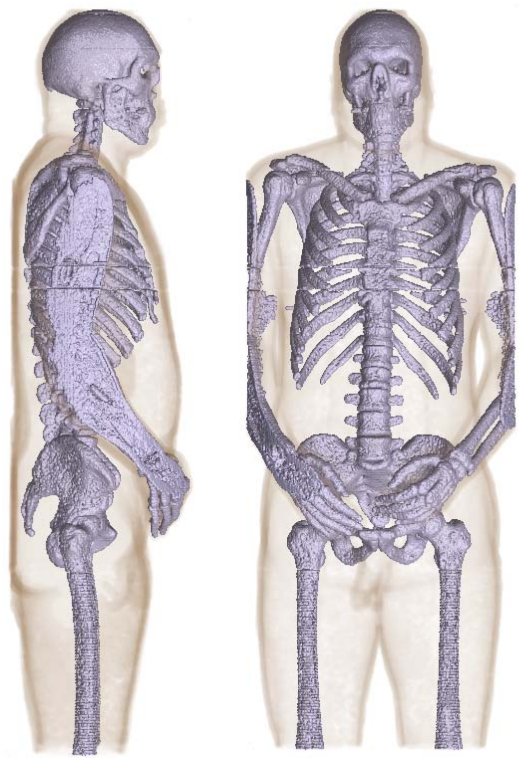

Figure 2: *Two Visible Human point sets, representing bones (1,218,973 points) and skin (267,303 points) respectively, are combined together using a volume scene graph. The point sets were part of the polygonal model provided by William E. Lorensen [Lor95] and made available by Georgia Institute of Technology.*

ume scene graphs. For example, it allows creation of combinational visualization of point sets and volume datasets. It opens the possibility of storing segments in a segmented volume as individual point sets (e.g., Figures 1 and 2). It enables the incorporation of advanced rendering features, such as shadows and refraction, which can add additional visual cues to a visualization (e.g., Figure 3).

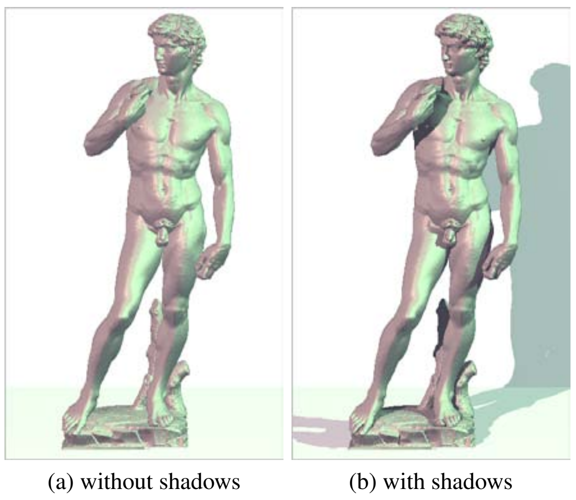

Figure 3: *The David dataset (Stanford) contains 28,184,526 points, for which an octree with 10 levels takes about 64 GB space. The visualization with shadows gives extra visual cues about the spatial relationship between the object and its surrounding, and between different parts of the object.*

As shown in [Che05], discrete ray tracing of a point cloud by brute force requires the evaluation of all points in the cloud, and it is hence not scalable in terms of the number of points. One can obtain significant speedup with a data partitioning scheme such as an octree. For example, tests in [Che05] indicated an average speedup at a factor of around 140 for a set of 10000 points. Therefore, memory becomes the fundamental bottleneck of ray tracing a volume scene graph with very large point sets, as it would demand an overwhelming amount of memory to accommodate all point datasets and their associated control structures.

In this work, we adopt octrees as the main data partitioning scheme as in [Che05]. Although the efficiency of such a data partitioning scheme is not the primary concern of this paper, there are good reasons for using this scheme in our example application. Firstly, it is one of the most commonly-used schemes in graphics and visualization. It is general enough for obtaining a fair comparison between different out-of-core algorithms, without introducing distortion due to some special algorithmic features, such as the assumption of opaque surfaces or pre-determined ray directions.

In this example application, each point dataset defines a *point-based volume object* (PBVO), where each point is associated with a radial basis function (RBF), and the RBFs are blended to form a scalar field of the PBVO. Like conventional volume objects, a PBVO can be associated with transfer functions and can be combined with other volume objects in a volume scene graph. The control structure of the entire volume scene graph is a hierarchical set of bounding boxes, in addition to the octree structure associated with every point set. Further details can be found in [Che05].

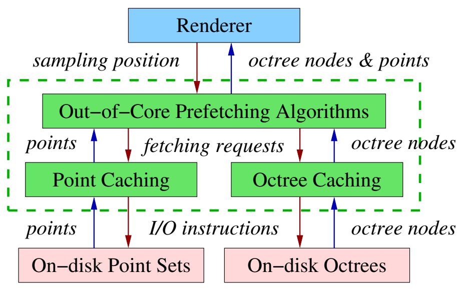

Figure 4: *The software architecture of the testing environment for our out-of-core algorithms.*

4. System Architecture

Figure 4 shows the overall architecture used for evaluating the out-of-core data management algorithms discussed in the next section. The renderer is a discrete ray tracer. The tested algorithms, which are contained in a *data management layer*, interact with the renderer through the same interface. The renderer informs the data management layer of the current sampling position s, and receives an octree leaf node which contains s if s hits a non-empty volume region. Each octree leaf node contains indexes of points which are stored in an out-of-core array. When the octree leaf node is loaded, all its associated data points are automatically loaded into the memory if they are not already in core.

The data management layer also includes two *out-of-core controllers*, for caching octree nodes and point data respectively. Three of the algorithms to be discussed issue prefetching instructions to the controllers, with an associated priority. If there is enough in-core space to fulfill these requests, the anticipated octree nodes (and the corresponding point data) will be loaded. When the in-core space is full, lower priority nodes are evicted to make space for higher priority ones. The priority of a node is set by the data management layer, according to the confidence of the prediction. The priority of all cached nodes is gradually decayed over time, except that whenever a node is referenced, it is set to the highest priority. This dynamic change of priority is very similar to the change of the scheduling priority of processes in UNIX operating systems.

5. Out-of-Core Data Management Algorithms

There are two main types of out-of-core data management algorithms, namely *demand-based* and *prediction-based*. The latter type is also referred to as *prefetching*, *pre-caching* or *anticipatory* algorithms. Similar to the notion used in operating systems, the *optimal algorithm* for out-of-core data management is an algorithm such that

- It would always prefetch the data block which will be required in the nearest future if it is not already in core.
- When the cache is full, it will always replace the data block that would not be required for the longest period of time.

The first condition applies to prediction-based algorithms only, while the second condition applies to both types of algorithms. Understandably, the optimal algorithm is very difficult, if not impossible, to realize.

The effectiveness of a prediction-based algorithm partially depends on its ‘knowledge’ of the application-specific logic. Many out-of-core algorithms are closely coupled with a particular application, for instance, surface extraction and volume ray casting. This allows some of the application-specific logic to be hard-coded into the out-of-core algorithms. However, such ‘knowledge’ is rather static, and such an algorithm usually makes predication in a rather ‘mechanical’ manner. An alternative approach is for the algorithm to acquire the ‘knowledge’ at run-time dynamically, and to adapt its prediction mechanism accordingly. In the following discussions, we refer those algorithms that acquire, store and use dynamic knowledge as *knowledge-based*.

A knowledge-based algorithm makes algorithmic decisions using inferred knowledge, and as the system runs, it learns how to function efficiently. In our case, the inferred knowledge relates to access patterns within the data to be visualized. A knowledge-based prefetching algorithm will learn the order in which data is accessed and automatically pre-cache data to be accessed next.

In the following subsections, we first describe a demand-based algorithm, which is used in Section 6 as a base-line for evaluation. We then describe three predication-based algorithms, starting with the *ray driven predictor* with the most hard-coded application-specific logic, followed by two knowledge-based algorithms, namely *access path predictor* and *history-based access predictor*.

5.1. The Least Recently Used (LRU) Strategy

This is a strategy commonly used in operating systems as a demand-based page replacement algorithm. It does not suffer from the so called Belady’s anomaly [SGG00], and provides a good approximation to the optimal algorithm without using pre-paging. We adapted this algorithm for managing the caching of octree nodes, and we implemented the algorithm in software by utilizing a queue that contains all cached octree nodes. The queue is organized in a temporal order. When an octree node is requested, if it is already in the cache, it will be moved to the head of the queue; otherwise, it is loaded into the cache and stored at the head of the queue. When the cache is full, the least recently used node, that is, the tail of the queue, is evicted to make room.

After the octree node is cached, the *out-of-core octree controller* informs the *out-of-core point set controller* of the point list from the accessed leaf node. On receipt of the point list, the *point set controller* checks that these are cached in-core, and if not, attempts to load them asynchronously.

5.2. The Ray Driven Predictor (RDP)

The RDP algorithm is closely coupled with our discrete ray-tracing engine used for rendering volume scene graphs. Given a ray and the current sampling position s_c, the algorithm makes an assumption that the octree nodes following the ray and along its path will likely be accessed subsequently. The algorithm determines a sampling point s_p on the ray which falls just outside of the current octree node. It then attempts to navigate the octree until it reaches the node that contains the predicted sampling point s_p. If, at any point on this navigation, it would need a node which is not already in the cache, it instructs the lower layer to prefetch it in preparation for the sampling at s_p. This process continues for each sampling point.

5.3. The Access Path Predictor (APP)

This is a knowledge-based algorithm, and it is aware of data related logic (i.e., the structure of the data), but only a very limited amount of rendering-related logic. It assumes that the most common access pattern within a dataset is the frequent appearance of the same *predecessor-current-successor* pattern. For example, when three octree nodes, N_a, N_b and N_c, are accessed in sequence, the pattern of $N_a \longrightarrow N_b \longrightarrow N_c$ can be recorded in N_b. When next time N_b is accessed, if N_a were the predecessor of N_b, we could use the recorded pattern $N_a \longrightarrow N_b \longrightarrow N_c$ to predict N_c as the next node to be required, and pre-cache N_c in advance.

Such an access pattern represents a section of a *path* along which an application navigates through its data. The APP algorithm particularly suits applications with more regular access patterns, such as ray tracing, marching cubes and feature tracking. However, unlike the RDP algorithm, it does not assume that N_a, N_b and N_c have to be on the same line, or N_a, N_b and N_c must be in the same neighborhood. Hence, it can accommodate a diverse range of rendering features such as refraction, space leaping, and so on.

In our implementation of the APP algorithm, four pairs of predecessor-successor are stored in each octree node, allowing the recording of up to four access patterns, each reflects a local view of some ‘access traffic’ passing through the node. Whenever a node is visited, its knowledge is updated. As the main costs of out-of-core management is related to the leaf nodes of an octree, we limit the acquisition and storage of knowledge to leaf nodes only. This restriction not only reduces a substantial amount of ineffective computational costs for knowledge acquisition, but also conveniently utilized the eight unused child addresses in each leaf node.

When the algorithm is unable to make a predication based on the recorded predecessor-current-successor patterns, it attempts to make a predication based on the four successors. If this is not successful, the algorithm resorts to the basic mechanism by navigating the octree from the current leaf node to a leaf node that contains the new sampling position. The algorithm is outlined in Algorithm 1.

Algorithm 1 The Access Path Predictor

```
Require: New sample point $s$,
Require: two recent octree nodes $N_{last}$, $N_{cur}$
  $N_{new} \leftarrow NULL$;
  if $s$ is in $N_{cur}$ then {If $s$ is in an cached node, do nothing}
    $N_{new} \leftarrow N_{cur}$;
  else {Phase I: Predication with Access Path}
    for each recorded predecessor $N_{p,i}$ of $N_{cur}$ do
      if $N_{last} == N_{p,i}$ and $s$ is in $N_{p,i}$ then
        $N_{new} \leftarrow N_{p,i}$;
      end if
    end for
  end if
  {Phase II: Predication with Historical Next}
  if $N_{new} = NULL$ then
    for each recorded successor $N_{s,i}$ of $N_{cur}$ do
      if $s$ is in $N_{s,i}$ then
        $N_{new} \leftarrow N_{s,i}$;
        return $Nnew$
      end if
    end for
  end if
  {Phase III: Fall back to octree navigation}
  if $N_{new} = NULL$ then
    $N_{new} \leftarrow$ OctreeNavigation($N_{cur}$, $s$)
  end if
  {Update knowledge in $N_{cur}$}
  Replace the oldest ($N_{p,i}$,$N_{s,i}$) in $N_{cur}$ with ($N_{last}$,$N_{new}$);
  {Precache octree nodes}
  if $N_{cur}$ is a recorded predecessor of $N_{new}$ then
    $N_{next} \leftarrow$ the corresponding sucessor in $N_{new}$;
    Precache $N_{next}$ with a high priority
  else
    Precache all four successors in $N_{new}$ with a low priority
  end if
  return $N_{new}$
```

5.4. The History-based Access Predictor (HAP)

The assumption of the frequent appearance of a certain pattern is a limited form of application-specific logic. We can remove this assumption completely, resulting in a knowledge-based algorithm that relies entirely on historic records.

For each leaf node, the HAP algorithm captures the knowledge of those nodes that were required immediately afterwards in previous visits to the node. It also utilizes the eight unused child address in each leaf node. When there are more than eight successors identified, it replaces the oldest successor recorded with the new one. When visiting a node, the algorithm pre-caches all known potential successors of the node. This potentially generates more disk traffic, since up to eight nodes are cached in order to load one.

5.5. General Remarks on APP and HAP

The two knowledge-based algorithms, APP and HAP, are aware of some of data-related logic, that is, the data partitioning mechanism, and the size of data blocks. The core part of both algorithms does not require the information about data connectivity, which is only used in the case the algorithms cannot make a prediction based on acquired knowledge. In general, they are not aware of the process being used to render a visualization image, and they infer knowledge based on existing access patterns. One minor exception is that APP utilizes an assumption about common traversal paths in order to improve the cost-effectiveness of knowledge acquisition, storage and reasoning.

Both APP and HAP mainly acquire the knowledge about leaf nodes, which link to the relevant point data. This means that the associated application frequently hops from one leaf node to another. In the absence of this knowledge, it would frequently be necessary to navigate several nodes up and then back down the octree, and each of these nodes could potentially require a disk access.

The secondary advantage of the ability to move directly between leaf nodes in an octree is that it reduces the size of the working set. There is a lower probability that non-leaf nodes will be required, and so more cache space can be used to store leaf nodes and associated point sets.

6. Results and Evaluation

We carried out a series of tests for evaluating the four algorithms described above, and compare the performance of the three prefetching algorithms with the LRU algorithms as a base-line benchmark.

In particular, we focused on two metrics, namely *cache hit rate* and *total disk reads*, which are commonly considered in operating system design. The cache hit rate indicates the percentage of requests which can be met without accessing the out-of-core storage. The total disk reads metric is the number of times that an out-of-core algorithm was required to load data from the disk. To facilitate comparison between rendering passes, the disk reads were normalized, using the LRU score for each pass as a base-line.

We first conducted a series of tests on randomly generated point sets of various sizes (between 100 and 100,000 points). Two types of point sets were generated in which the points were placed randomly on a spherical surface, and within a spherical volume. Tables 1 and 2 include a summary of the results for of the tests. Figure 5 shows the normalized disk reads for the random point sets comprising points in a spherical volume. Figure 6 shows the cache hit rates for the same data sets. Each rendering pass was conducted at three different sampling intervals, which represent a type of rendering-related logic.

From these results we can observe that the RDP and the HAP show a significant increase in the number of disk reads required, while APP performs better even than LRU by this metric in spite of performing pre-caching. All three prefetching algorithms achieved higher hit rates than LRU, with marginal variation.

Table 1: *Summary of normalized disk reads for random data, averaging varying data sizes and sampling distances.*

Dataset	LRU	RDP	APP	HAP
Points on a Surface	1	1.81	0.81	1.92
Points in a Volume	1	2.20	0.87	2.44

Table 2: *Summary of cache hit rates for random data.*

Dataset	LRU	RDP	APP	HAP
Points on a Surface	98.90	99.83	99.74	99.76
Points in a Volume	99.66	99.85	99.84	99.87

Figure 5 shows that the performance of APP is consistent over a variety of point sets and sampling intervals, while the performance of the other prefetching algorithms varies considerably. Figure 6 shows that the performance of LRU is adversely affected by the increase in data size. It is also affected more greatly by coarse sampling intervals due to the poor locality of reference here, clearly indicating the advantage of prefetching.

We also tested the four algorithms in relation to a number of real data sets, including the procedurally generated hyperbolic surfaces (Figure 1), a single Stanford Bunny, a scene containing twenty bunnies (see [Che05]) and two Visible Human point sets for skin and bone respectively (Figure 2). For the scene with twenty bunnies, we purposely loaded 10 bunny datasets (two bunnies per dataset) in order to evaluate the performance in relation to multiple point sets in a volume scene graphs. Some of the tests (e.g., the hyperbolic field with 5 million points in Figure 1(d)) would almost halt an average desktop computer without out-of-core data management. Tables 3 and 4 summarize the normalized disk reads and cache hit rates for these data sets. Figures 7 and 8 show more detailed results with different sampling intervals.

As we can see from Table 3, APP gives performance comparable to LRU, while the other two prefetching algorithms result in an increased disk load. Table 4 shows that APP's better performance in terms of disk reads comes at a slight expense in terms of hit rates, but still delivers better performance than a demand-driven LRU strategy.

RDP, the algorithm with the most hard-coded knowledge, performed consistently better than the other three, being the only algorithm to achieve a hit rate of over 99% in all tests. The two knowledge based algorithms, however, improved the cache hit rate by over 1%.

Note that although this improvement of 1% seems small, Amdahl's law [Amd67] shows us that this can generate a significant performance improvement. Each failed cache hit stalls the rendering algorithm until the data can be loaded

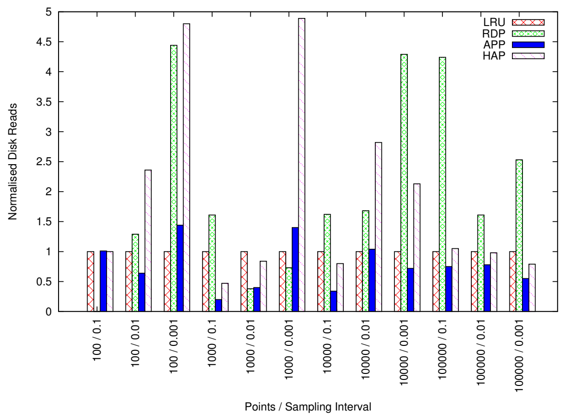

Figure 5: *Normalized disk reads for random data sets comprising points in a spherical volume.*

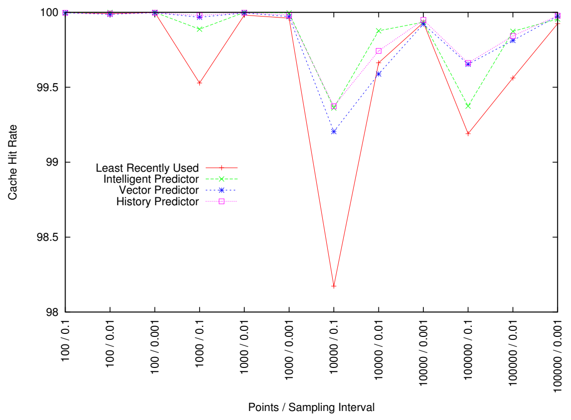

Figure 6: *Cache hit rates for random data sets comprising points in a spherical volume.*

from disk. Since a disk reads usually takes around 9ms, which equates to 9,000,000 instruction cycles lost on a moderately fast computer.

On the other hand, although a 30% increase in disk reads may seem a lot, it does not have the same impact as failed catch hits. Because of asynchronous disk reads, the disk accesses activated by the prefetching algorithm do not stall the rendering algorithm, and normally run in parallel with the rendering task. Hence, in terms of overall rendering speeds, the impact of disk reads caused by prefetching is generally not noticeable.

In general, the improvement of cache hit rate translates to an improvement in frame-rate, especially for vary large datasets rendered with highly constrained in-core cache size. For example, consider the rendering of the David dataset (Figure 3) on a low-specification computer with a 1GB RAM and 1GB swap space and a 1.4GHz Athlon processor. As the point dataset, together with opacity and radius of each RBF, requires over 0.5GB space, and its octree (of height 8)

Table 3: *Summary of normalized disk reads for real data.*

Dataset	LRU	RDP	APP	HAP
Stanford Bunny	1	1.54	1.21	1.57
Twenty Bunnies	1	1.54	1.17	1.63
Hyperboloid	1	1.24	0.94	1.07
VH Bone	1	1.06	0.77	1.05
VH Skin	1	1.14	0.95	1.08
Average	1	1.30	1.01	1.28

Table 4: *Summary of cache hit rates for real data.*

Dataset	LRU	RDP	APP	HAP
Stanford Bunny	96.19%	99.29%	97.67%	98.19%
Twenty Bunnies	96.19%	99.29%	97.22%	98.19%
Hyperboloid	97.83%	99.55%	99.63%	99.68%
VH Bone	99.54%	99.99%	99.95%	99.95%
VH Skin	99.79%	99.99%	99.97%	99.97%
Average	97.91%	99.62%	98.89%	99.20%

takes 1.96GB, the computer cannot support the discrete ray tracer without out-of-core data management. We applied the above-mentioned algorithms to the rendering of the David dataset, with an additional constraint of 14MB virtual memory space. On average, we saw a 10-15% speed improvement using HAP over LRU.

7. Conclusions

In this paper we have presented two new knowledge-based algorithms, APP and HAP, for out-of-core data management in visualization. In conjunction with an example application, we have shown that a knowledge-based approach can provide cache hit rates of a similar quality with, and often exceeding an algorithm designed using hard-coded application-specific logic. In the case of APP, this improvement does not come at the expense of increased disk load, though its assumption of predecessor-current-successor patterns may not be effective for all visualization applications. On the other hand, HAP is more flexible, but requires relatively high throughput in disk access.

Most existing prefetching algorithms speculatively cache a large amount of data that is never accessed, imposing a significant load on the storage device responsible for the data set. We have shown that a knowledge-based approach does not necessarily need to carry this penalty, and have presented a novel algorithm, APP, which provides a good trade-off between disk accesses and cache hit rates.

Our results indicate that it is possible to design a generic knowledge-based out-of-core algorithm that can support a variety of visualization applications. This opens the possibility of designing and developing a standard API that contains a collection of out-of-core data management algorithms which will be the focus of our future work.

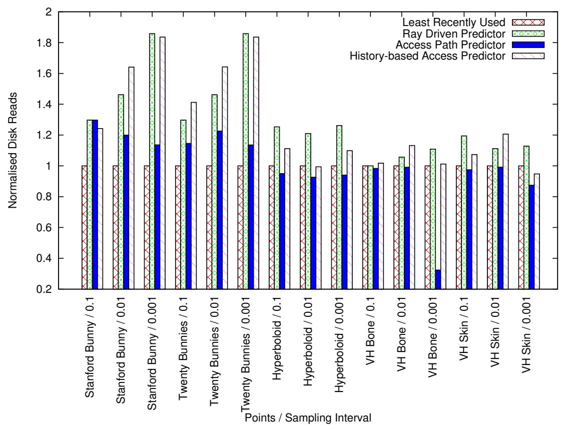

Figure 7: *Normalized disk reads for real data.*

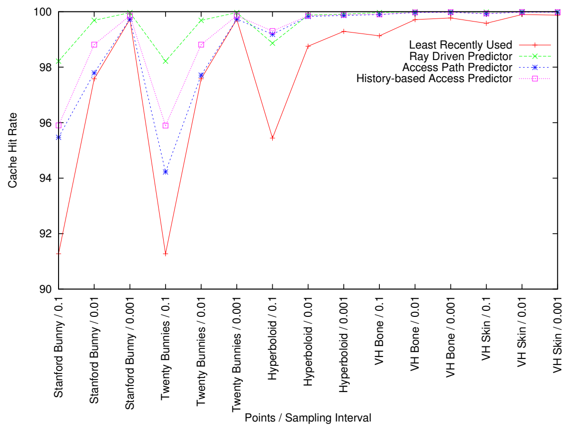

Figure 8: *Cache hit rates for real data.*

References

[Amd67] AMDAHL G. M.: Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. AFIPS Spring Joint Computer Conference* (1967), pp. 483–485. 6

[CE97] COX M., ELLSWORTH D.: Application-controlled demand paging for out-of-core visualization. In *Proc. IEEE Visualization* (1997). 2

[CFSW03] CHIANG Y.-J., FARIAS R., SILVA C. T., WEI B.: A unified infrastructure for parallel out-of-core isosurface extraction and volume rendering of unstructured grids. In *Proc. IEEE Symposium on Parallel Visualization and Graphics* (2003). 2

[Che05] CHEN M.: Combining point clouds and volume objects in volume scene graphs. In *Proc. Volume Graphics* (New York, 2005). 2, 3, 6

[CMPS96] CIGNONI P., MONTANI C., PUPPO E., SCOPIGNO R.: Optimal isosurface extraction from irregular volume data. In *Proc. IEEE Symposium on Volume Visualization* (1996), pp. 31–38. 1, 2

[CS97] CHIANG Y.-J., SILVA C. T.: I/o optimal isosurface extraction. In *Proc. IEEE Visualization* (1997), pp. 293–300. 2

[CS98] CHIANG Y.-J., SILVA C. T.: Interactive isosurface extraction. In *Proc. IEEE Visualization* (1998), pp. 167–174.

[DC03] DANIEL G. W., CHEN M.: Video visualization. In *Proc. IEEE Visualization* (2003), pp. 409–416. 1

[Den68] DENNING P. J.: The working set model for program behaviour. *Communications of the ACM 11*, 5 (May 1968), 323–333. 2

[Den70] DENNING P. J.: Virtual memory. *ACM Computing Surveys (CSUR) 2*, 3 (Semptember 1970), 153–189. 2

[FS01] FARIAS R., SILVA C. T.: Out-of-core rendering of large, unstructured grids. *IEEE Computer Graphics and Applications 21*, 4 (2001), 42–50. 1, 2

[Kou99] KOUTSOFIO E. E.: Visualizing large-scale telecommunication networks and services. In *Proc. IEEE Visualization* (1999). 1

[Lin00] LINDSTROM P.: Out-of-core simplification of large polygonal models. In *Proc. SIGGRAPH* (2000), pp. 259–262. 2

[LM99] LEUTENEGGER P., MA K.-L.: Fast retrieval of disk-resident unstructured volume data for visualization. In *External Memory Algorithms and Visualization (DIMACS Book Series, American Mathematical Society)* (1999), vol. 50.

[Lor95] LORENSEN W. E.: Marching through the visible man. In *Proc. Visualization* (1995), pp. 368–373. 3

[LP02] LINDSTROM P., PASCUCCI V.: Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 239–254. 2

[PKGH97] PHARR M., KOLB C., GERSHBEIN R., HANRAHAN P.: Rendering complex scenes with memory-coherent ray tracing. In *Proc. SIGGRAPH* (1997), pp. 101–108. 2

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: surface elements as rendering primitives. In *Computer Graphics (Proc. SIGGRAPH 2000)* (2000), pp. 335–342. 2

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: a multiresolution point rendering system for large meshes. In *Proc. SIGGRAPH* (2000), pp. 343–252. 2

[SCESL02] SILVA C. T., CHIANG Y. J., EL-SANA J., LINDSTROM P.: Out-of-core algorithms for scientific visualization and computer graphics. In *Proc. Visualization* (2002). 2

[SCM99] SHEN H.-W., CHIANG L.-J., MA K.-L.: A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proc. IEEE Visualization* (1999), pp. 371–378. 1, 2

[SGG00] SILBERSCHATZ A., GALVIN P., GAGNE G.: *Applied Operating Ssytems Concepts*. John Wiley & Sons, 2000. 2, 4

[SH00] SUTTON P. M., HANSEN C. D.: Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (2000), 98–107. 2

[TFFH94] TELLER S., FOWLER C., FUNKHOUSE T., HANRAHAN P.: Partitioning and ordering large radiosity computations. In *Proc. SIGGRAPH* (1994), pp. 443–450. 2

[USM97] UENG S.-K., SIKORSKI C., MA K.-L.: Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics 3*, 4 (1997), 370–380. 2

[Vit01] VITTER J. S.: External memory algorithms and data structures: dealing with massive data. *ACM Computer Survey 33*, 2 (2001), 209–271. 1, 2

[VM02] VARADHAN G., MANOCHA D.: Out-of-core rendering of massive geometric datasets. In *Proc. IEEE Visualization* (2002). 2