methods were deemed unsuitable for remote visualization because the large size of the IBR database they generated could not be efficiently communicated over the Internet using conventional TCP socket calls.

There are several reasons for seeking to use light field methods for remote visualization. First, the rendering process of a light field database is simply a sequence of table lookup operations, enabling the use of client devices, such as PDAs, that lack even graphics acceleration. Second, the resource requirements for this method scale directly with the pixel resolution on the user console, which provides a natural tradeoff between visualization needs and resource availability. Third, unlike many existing remote visualization approaches, it is capable of handling the most general form of volume rendering with both semi-transparency and full opaqueness. The number of sample views used to generate the light field database correlates well with the achievable accuracy in the final renderings on the client display. We have been able to interactively browse through a light field database of multiple gigabytes stored in California at more than 30 frames per second on the campus of University of Tennessee, Knoxville. We briefly introduce the background of our work in Section 2, and we then discuss the details of our prototype system and present results we achieved in Section 3 and 4, respectively.

2. Background

To properly introduce the background of our work, we divide this section into two parts, discussing the general background of remote visualization and the Logistical Networking infrastructure separately.

2.1 Remote Visualization

Remote visualization often refers to scenarios in which the volume dataset is stored on a server but the visualization is needed on a remote client console. A session of remote visualization requires runtime collaboration between the server and the client(s) across the Internet. The simplest form of remote visualization involves transmitting images rendered on the server to the client, where user input requesting new images is collected and sent back to the server. To achieve high interaction rates and low latency, a dedicated telecom trunk, like an ATM link, has often been used [10]. This approach also assumes that a volume dataset can always be rendered at real time frame rates. Unfortunately, most routine scientific datasets require minutes to be fully rendered under any given view. Hence, this brute force approach is not applicable to general usage.

Due to the limitation on network resources, most remote visualization methods proposed for wide area environments provide the client with some type of graphical representation which is sufficient to support local user interaction for a window of views and make per-frame update from the server unnecessary. A server update is only needed when the user navigates outside of the supported window. Examples of such graphical representations include iso-surfaces [17], view dependent iso-surfaces [11] and view dependent texture-mapped depth meshes [5, 12, 13]. Extracting the iso-surface of a data value from a volume produces a much-simplified version of the original dataset [17]. If the extracted iso-surface can be fully stored on the client, then the client can mostly function by itself and a new update from the server is only necessary when the iso-value is to be changed. However, if the full iso-surface is still too big for the user console, or the iso-value needs to be frequently manipulated, extracting and transmitting view dependent iso-surfaces from the server is more viable [11]. In this case, when the user moves outside the range of supported view angles, a new set of view-dependent iso-surfaces is needed. A common limitation of all iso-surface methods is the lack of ability to render any volumetric effects present in the volume dataset.

To capture volumetric effects, Mueller et. al. [13] and Bethel et. al. [5] pioneered partitioning a volume into depth slabs. When each slab is volume rendered, the resulting color buffer is recorded as a texture, while the depth buffer is used to construct a coarse geometry mesh. During rendering on the user console, these coarse depth meshes are rendered in 3D with textures mapped to show realistic effects of 3D parallax. Interestingly, such a simple representation has been shown to support a rather considerable neighborhood of views around the view under which the representation is generated. By caching such a representation locally, rather intuitive remote visualization of a volume dataset can be obtained as long as the server can provide a new set before the user movement steps outside the supported local neighborhood. Later, Luke and Hansen [12] devised a similar method where the depth mesh is used for more realistic lighting computation on the user console. But unfortunately, for all such texture aided mesh approaches, the absence of visual artifacts in those visual interactions does not necessarily mean accuracy in rendering. It is quite difficult to gauge the correctness of a view rendered on the user console during live interaction, and this tends to degrade user confidence.

Following a different line, tools have also been developed using large-scale parallel networking, data storage and computation to provide high-end visualization via ultra-high performance Grid computing infrastructures [6, 7]. The target community for such approaches is quite different from ours, however. Instead of supporting high-end applications and users, we would like to devise a more scalable and affordable approach to support low-end users and devices. Our intended remote visualization system should meet the following criteria: It should (i) require a low amount of computation on the user console so that even lightweight devices are supported; (ii) consume system resources in amounts that are scalable to the level of visualization needed on the user console, as measured by desired pixel resolution and interactive rates, and (iii) support general viewing situations (opaque or semi-

transparent) with a direct metric of correctness in the images displayed on the user console. No existing approach currently meets all three goals. Here, we explore light fields [8, 9], a form of IBR, to meet these goals. The reason that light field has previously not been widely considered for remote visualization is the immense size of the IBR databases generated. By leveraging the power of the Logistical Networking infrastructure to provide higher network transmission rates and more flexible caching close to the client, we believe it is possible to use light fields for remote visualization. The infrastructure of LoN is discussed in Section 2.2, and we discuss the background of the original proposal of light field rendering as well as how we adapted it for remote visualization purposes in Section 3.

2.2 **Logistical Networking (LoN)**

The concept of Logistical Networking is based upon an exposed approach to computer service architecture, in which one offers client software a primitive service whose semantics are closely based on the underlying physical infrastructure [3]. Higher-level tools and protocols with more abstract semantics running on clients compose these primitive services to implement more sophisticated run-time algorithms for applications. In this way the infrastructure of LoN follows the exposed approach embodied in IP datagram service. But in some respects, LoN represents a more exposed approach. LoN uses scalable storage resources in the network to model communication in *both* its synchronous and asynchronous aspects [2]; its fundamental service is the movement of data between buffers on adjacent nodes.

| Application |
| Logistical File System |
| Logistical Runtime System (LoRS) |
| exNode | L-Bone |
| IBP |
| Local Access |
| Physical |

Figure 1. Network Storage Stack

The Internet Backplane Protocol (IBP) supplies LoN with its basic mechanism. It implements scalable management of storage in the network using shared physical resources [4, 15]. IBP is implemented by intermediate nodes, called "depots," that offer standard storage operations, including allocate, load, store and third party copy. From a networking perspective, an IBP depot can be viewed as a kind of router that exposes buffers to clients, thereby enabling the implementation of flexible network services, such as overlay multicast with explicit specification of the delivery tree by the source.

As the name suggests, the goal of Logistical Networking is to bring data transmission and storage within one framework, much as military or industrial logistics treat transportation lines and storage depots as coordinate elements of one infrastructure. Achieving this goal requires a form of network storage that is globally scalable, meaning that the storage systems attached to the global data transmission network can expand beyond organizational and geographical boundaries and can grow large enough to be global in scope.

To create a shared resource fabric that exposes network storage for general use in this way, Beck, Moore and Plank defined a new *storage stack* (Figure 1), analogous to the Internet stack, using a bottom-up and layered design approach that adheres to the end-to-end principles [15]. The important thing to note here is that the key to achieving scalability using this model lies in defining the right basic abstraction of the physical resource to be shared at the lowest levels of the stack. For Logistical Networking IBP plays this role.

IBP is the lowest layer of the storage stack that is globally accessible from the network. Its design is modeled on the design of IP datagram delivery. Just as IP is a more abstract service based on link-layer datagram delivery, so IBP is a more abstract service based on blocks of data (on disk, memory, tape or other media) that are managed as "byte arrays." By masking the details of the storage at the local level — fixed block size, differing failure modes, local addressing schemes — this byte array abstraction allows a uniform IBP model to be applied to storage resources generally. The use of IP networking to access IBP storage resources creates a globally accessible storage service.

As the case of IP shows, however, in order to scale globally the service guarantees that IBP offers must be weakened, i.e. it must present a "best effort" storage service. First and foremost, this means that IBP storage allocations can be time limited. When the lease on an IBP allocation expires, the storage resource can be reused and all data structures associated with it can be deleted. Additionally an IBP allocation can be refused by a storage resource in response to over-allocation, much as routers can drop packets, and such "admission decisions" can be based on both size and duration. Enforcing time limits puts transience into storage allocation, giving it some of the fluidity of datagram forwarding; more importantly, it makes network storage far more sharable, and creates an infrastructure that can scale.

The semantics of IBP storage allocation also assume that an IBP storage resource can be transiently unavailable. Since the user of remote storage resources depends on so many uncontrolled, remote variables, it may be necessary to assume that storage can be permanently lost. Thus, IBP is a "best effort" storage service. To encourage the sharing of idle resources, IBP even supports "soft" storage allocation semantics, where allocated storage can be revoked at any time. In all cases such weak semantics mean that the level of service must be characterized statistically.
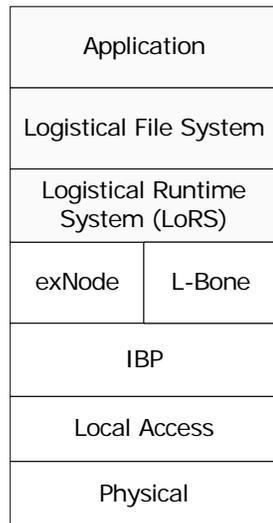
IBP storage resources are managed by "depots," which are servers on which clients perform remote storage operations. A detailed account of the API and its other functions is available in [15]. In [14], multi-threaded algorithms for high-performance downloads of wide-area, replicated data are presented. These IBP download methods have achieved network data transfers rates over 100Mb/s in high speed research networks such as Abilene and ESNet.

As already noted, one good way to attack the problem of network latency in remote visualization applications is to use some form of intermediate network buffer under application control. LoN depots, which provide a generic storage service exposed for application scheduling, are a natural choice for this purpose. By using depots to pre-distribute the datasets into the Internet and then automatically streaming the data to the client from the closest depot(s), the majority of overhead for storage management is offloaded from both the server and the client. The *Logistical Backbone* (L-Bone) allows the user to find the closest set of IBP depots that can satisfy the needs of an application. We use the L-Bone tools to dynamically identify appropriate depots to serve as the network caches.

The exNode is an XML-encoded data structure for aggregation of capabilities, or strings that specify different parts of a dataset stored in different storage locations represented by each of the capabilities. ExNodes are modeled on the inodes that are a familiar part of the Unix file system, except that exNodes map the data extent of a file into IBP allocations on depots rather than to blocks on a local disk. The client requests and retrieves the IBR data from the network on demand using the exNode. Using IBP and exNodes, our streaming model puts large-scale datasets into the network and only needs to cache the exNodes at the client, allowing multiple depots to be used simultaneously, and probably with data replication.

3. Our System

3.1 Light Field Rendering

As the complexity of the geometric datasets explodes to capture as many details as possible, often several triangles or voxels are projected into the same pixel. Under these circumstances, to continue to model a scene with raw geometries seems to be too inefficient. Hence, image based rendering (IBR) was proposed to directly model a scene as imagery. Light field rendering [9] is one of the methods proposed within the framework of IBR.

One way to model a scene with imagery is to pre-capture the color of all possible viewing rays when the dataset is rendered at arbitrary views, which can be parameterized with seven variables, i.e. the starting location of a viewing ray (x,y,z), the direction of the viewing ray (theta, phi), the wavelength (lamda) and the instant time (t). Borrowed from the word, plenum, such a 7-dimensional space is refered to as a plenoptic function. The true continuous analytical forms of plenoptic functions are generally unknown. Only discrete sampled versions of plenoptic functions can be captured digitally. However, this results in an overwhelming need of storage [9]. To address this issue, light field reduces the plenoptic function to 4-dimensions by omitting time and wavelength, and by constraining camera locations to the outsize of the bounding box around the dataset. In this case, whenever the viewer is behind a boundary plane outside the bounding box, all possible viewing rays that may intersect the dataset can be described by four parameters. All one needs is two planes, P_1 and P_2, parallel to the boundary plane. Any viewing ray intersecting the bounding box of the dataset must intersect P_1 and P_2 as well. The two intersection points on P_1 and P_2 can be parameterized with (s,t) and (u,v), respectively.

Using a lattice of cameras, the light field database is easily captured. In this case, P_1 is the focal plane of the cameras and P_2 is the plane on which the camera lattice is located. Oftentimes, one only needs to render an image on each location in the camera lattice; these images are referred to as *sample views*. The size of the light field database only depends on the number of sample views taken and the pixel resolution of each sample view. In this aspect, the complexity of the IBR database is detached from the complexity of the underlying volume dataset. Of course, proper low-pass filtering, or anti-aliasing, is necessary if the pixel resolution is lower than the Nyquist rate required by the dataset. To render a novel view from the light field database, the color of each viewing ray is looked up directly from the light field database according to (s,t,u,v). When the value of a pixel is not already stored in the discrete 4D space, interpolation is necessary.

Often times, to faithfully capture the appearance of a complex dataset, tens of gigabytes are needed to store the discrete 4D ray space. Compression techniques can be used to reduce the size of data. However, lossy compression at high compression rates can degrade the visual quality of final rendering significantly.

3.2 Light Field Database for Remote Visualization

The planar parameterization used in the original light field method requires camera locations to be behind a boundary plane. This may cause undesired complexity when full 3D user interactions, which involve panning and zooming, are needed. In our work, we extend the original light field method to use spherical parameterization instead. We still assume a volume to be examined only from outside and put two concentric spheres around the volume. No matter where a viewpoint is located outside the spheres, any possible viewing ray through the volume penetrates the two spheres. The resulting two intersection points can both be parameterized by the angular components in spherical coordinates, (θ,ϕ). To stay with the current convention in light field rendering, we still use (s,t) and (u,v) as the symbols for these points. A viewing ray through the volume

is still indexed by (s,t,u,v) in the 4D ray space. A light field database so constructed can only support "replaying" the external views of a volume. To allow user navigation through the interior of a volume, multiple light field databases are needed [16], but the same framework for remote visualization can be reused.

Spherical parameterization differs from planar parameterization in that not all (s,t,u,v) combinations are valid, due to occlusion. Leveraging this fact, we can naturally save storage by not storing portions of the 4D database that will remain empty. We use the outside sphere as the location of the camera lattice of $m \times n$ locations, with the resolution of the image rendered at each camera location being $r \times r$. This process provides us with a 4D database of $m \times n \times r \times r$ values. Note that $r \times r$ is much smaller than the inherent resolution of (s,t) on the inside parameter sphere. We further organize the $m \times n \times r \times r$ database into *view sets*, by partitioning the $m \times n$ lattice into groups of size $l \times l$. A view set provides a natural mechanism to exploit view coherence and is the smallest unit of network transmission we use.

In Figure 2, for instance, we have positioned two concentric spheres around a volume block. There are two view sets shown in different colors. In our current implementation, we use sample views at an interval of 2.5 degrees, requiring a 72×144 camera

Figure 2. An illustration of view set partitions.

lattice. We use an *l* value of 6, so that each view set covers a view angle range of 15 degrees. Accordingly, there are 12×24 view sets in the whole database. In order to support user interaction, the user console only needs to have the view set that encompasses the current view angle. As long as a new view set is delivered to the user console before the user moves outside the 15 degree window handled by the current view set, network latency will not affect local user interaction. Our prefetching mechanism developed for this purpose is discussed in Section 3.5.
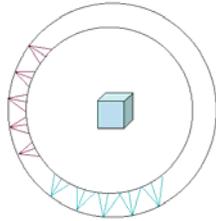
The $r \times r$ resolution should roughly correspond to the desired viewing resolution on the user console. Since most pixel content is not shown on the user display anyway, too high a pixel resolution makes unnecessarily high demands on storage and network resources. A design issue exists, however, when a user zooms into the dataset for close-up views to examine physical details. Because such movement is often localized and could be predicted with high probability, and those x a very small portion of the whole volume, it is feasible for the corresponding view set to be generated on the fly.

3.3 The Streaming Model

In this section, we describe the streaming model we designed to deliver the light field database. The LoN infrastructure provides the core technical support for this approach, so we call our system the LoN Enabled Browser of Image Based Database. The modules involved in the streaming model (Figure 3) include: server and server agent, depot pool and Dictionary of View Sets (DVS), client and client agent. A key algorithm for guaranteeing a satisfactory level of performance is the one governing network caching and prefetching of view sets driven by user interaction.
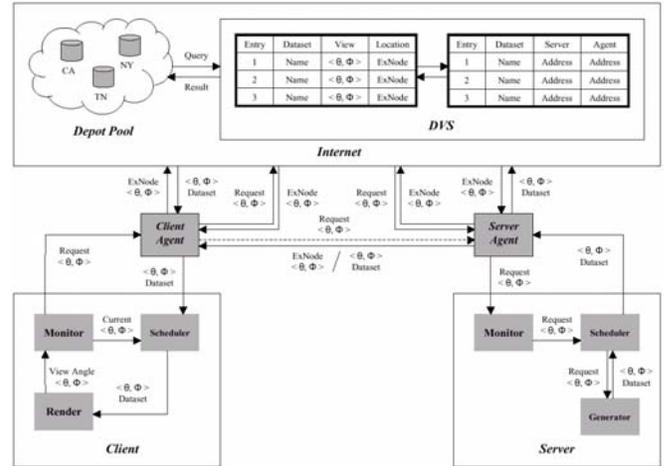
Figure 3: Streaming model over LoN infrastructure

Before we discuss the details of each module, we illustrate (Figure 3) the general pipeline of workflow in the system. The server generates the light field database, i.e. view sets, which are then uploaded to IBP depots distributed across the Internet. Although these depots are not part of the server, they assume the role of serving view sets to client agents. Hence, in our system, these depots are referred to as *server depots*. The same view set may be replicated on multiple server depots. As in other IBP applications, the stored view sets are accessed via a corresponding exNode. A dictionary service listing each view set, as well as its exNode pointer and other auxiliary information, is created and maintained on a server of a Dictionary of View Sets (DVS server). The client supports user navigation. As a new view set is needed, the DVS server is contacted with a query for the exNode of this view set. The retrieved exNode is then used to obtain a copy of the view set, via IBP downloads. A client agent brokers the communication from client to all other modules. The client agent is a separate computing process that may or may not be on the same computer where the client is run. The server also has a server agent that may likewise be separate.

3.4 Server and Server Agent

The generator in the server renders the volume datasets into view sets. In our system, the generator also compresses each view set with the lossless scheme zlib [1].

The rendering of all view sets can be completely pre-computed off-line. Alternatively, at run time, the client agent can dynamically query the server for view sets that have not been rendered, as for instance when the user is zooming in for a close-up view of a very small portion of the volume. Our generator uses a parallel ray-caster on 32 processors for rendering. The server monitor is the

Figure 4. An illustration of the prefetching policy.

interface for all such run-time queries for view sets that have not been already rendered. Working from the entire collection of requests that have been received but not yet rendered, the scheduler chooses the latest request to assign to the generator. After the generator renders a view set, per request of the scheduler, a copy is sent to the client agent and the pool of server depots, and the DVS is updated.
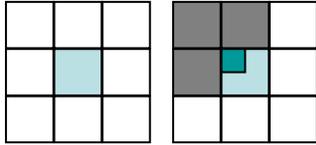
Using a server agent seperate from the server offers flexibility of permitting multiple servers. Such flexibility may be necessary when the simulation dataset itself is not stored at a single geographical location. The server agent hides such details from other parts of the system.

3.5 Client and Client Agent

The client process appears as the graphical interface interacting with the user. It takes user input and renders the desired view, if that view is within the current view set that is locally stored. Otherwise, it asks the client agent to request new view sets and waits for the agent to update it. The view sets received by the client are then decompressed. Since the client agent handles communication and caching on behalf of the client, the client only requires a low amount of computing and storage capability.

A client agent can serve multiple clients, especially in a mobile environment, in which a network cache dedicated for an individual client is difficult to implement. If it receives a request for a view set it has already cached, it sends the cached view set directly to the requesting client. If it hasn't cached the actual view set but only the exNode for the requested view set, it uses that exNode to download the view set before delivering to the client. When the requested view set is completely new to the client agent, the agent queries the DVS server for the exNode pointers, and then either gets the view set directly from the server agent (if the server needs to render it at runtime), or gets the exNode from the network and then downloads the view set. To accelerate the dataflow to the client, the client agent maintains a cache of both view sets and the exNodes of view sets recently downloaded or pre-fetched. In order to serve its caching and streaming purposes, the client agent needs to be equipped with faster computation and memory units, as well as high-bandwidth Internet connections. In the

current IP networking model, the analogue of our client agent is the local area router.

In order to hide part of the network latency, the client agent needs to prefetch view sets that may be needed in near future. Our initial prefetch policy is illustrated in Figure 4. The left diagram shows a center view set and its eight neighbors. When the user navigates within the span of the central view set, only a subset of the eight neighboring view sets is probably needed, depending on which quadrant of the span the user is in. For instance, when the user is within the top-left quadrant (deep blue) of the central view set, only the neighboring view sets to the top and left of the central view set (dark gray) may be needed. Based on this straightforward estimation of possible user movement outside the central view set, we prefetch those view sets that may be needed but are not yet locally available.

As we describe in Section 4, we have improved our prefetching results through the use of an IBP depot in the local area network of the client agent, which is used for very aggressive prefetching of view sets. The use of an intermediate depot offloads prefetching transfers from the client agent, reducing the cost of extraneous prefetch almost to zero. The client agent fills its own cache from the local depot, if possible, and from the remote server depot, when necessary. What differentiates LoN from other forms of storage management for such uses is precisely the open availability of substantial depot resources — storage, memory, processor, and bandwidth — that can be shared by applications within both the local and wide area network.

Aggressive staging of all viewsets to a local depot, ordered by proximity to cursor.

Staged viewsets available in LAN

When cursor movement triggers prefetch of viewsets to the client agent, they are available locally.
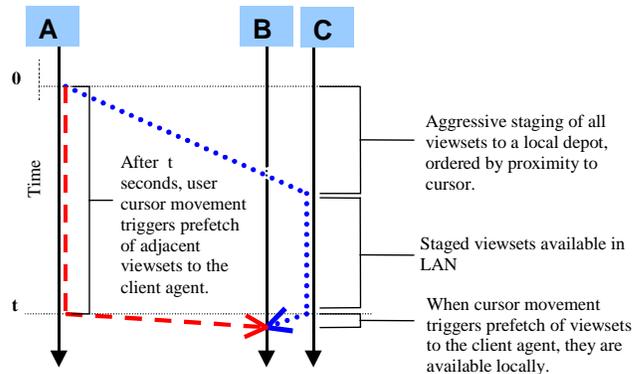
Figure 5: Aggressive two-stage prefetching. A is the server depot, while B is the client agent. C is a depot on the local area network of the client agent, B.

In particular, our aggressive prefetching algorithm works in two stages (Figure 5). Conventional prefetch (red arrow) is driven by user movements and, hence, often spontaneous in nature. Besides extraneous prefetches, the latency incurred by data transfer in the WAN may not be fully smoothed out, and this will affect the quality of user interaction. This may happen whenever the resources available in the best-effort IP network become scarce. With LoN, we can utilize an intermediate depot on the local area network of the client agent, frequently leveraging depot

6

resources that have already been deployed to serve other applications. While the network is vacant, aggressive staging of view sets that may be soon requested are performed, exploiting every bit of available network bandwidth. The longer the network remains vacant, the more view sets are staged. All such LoN operations take place as third party communication without consuming resources on either the client or the client agent. When a new prefetch request is initiated, we directly query the view sets cached on the local depot, bypassing the relatively slower wide area network. Thus, the poor performance of the WAN is much more likely to be addressed.

3.6 Dictionary of View Sets (DVS)

The DVS server must be able to service a large number of queries at the same time. It maintains two types of look-up tables: the (i) exNode table and the (ii) server agent table. The exNode table is indexed by identifiers of view sets and stores the corresponding exNode pointers. Note that each view set may have multiple exNodes, pointing to multiple replicas of that view set on the network. When a request is received, it looks up the exNode table. If it finds the requested view set, it returns the corresponding exNode for the client agent to download. If it doesn't, the DVS resorts to the second lookup table, the server agent table, where the records show which server agent needs to be contacted. The DVS then forwards the request to the right server agent for generation and uploading of the view set at runtime. It updates the exNode table with the exNode returned by the server agent.

In view of the large size of exNode tables, the DVS server is implemented in a hierarchical fashion for efficient queries. Any query will go through all levels recursively until the request is fulfilled. If the requested dataset cannot be found when the lowest level in the DVS has been reached, the requested view set has not yet been computed. In some respects, the DVS service in our system is quite similar to the Domain Name Service (DNS) in the Internet.

4. Results and Discussions

4.1 Construction of Light Field Database

We construct the light field database (LFD) on a cluster of 2.4 GHz Pentium 4 processors, connected by Dolphin Networks. The scientific dataset we used for testing purposes is a simulation of electrical potential of a negative high-energy protein (negHip). The volume itself, at $64 \times 64 \times 64$, is not large. While this is a small volume dataset, we would argue that the complexity of light field datasets is detached from the resolution of the volume dataset. Using the negHip dataset, we can still safely make sound analysis of how our prototype system performs. In Figure 6, we show two screenshots from the monitor of the client console during a session of remote visualization.
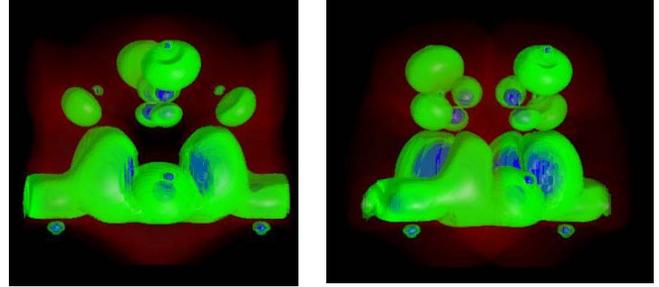

Figure 6. Two screen captures on the client console during a remote visualization experiment exploring the negHip dataset.

In the LFD, the sample views are spread on a lattice, with intervals being 2.5 degrees in spherical angles. We use view sets covering 15 degrees in each angular component. The uncompressed size of the total database ranges from 1.5GB to 14GB, at image resolutions in each sample view of 200×200 to 600×600 (Figure 7). Using loss-less zlib compression, we achieved 5 to 7 times compression rates, at image resolutions ranging from 200×200 to 600×600. The maximum total size of compressed light field database is around 2GB.
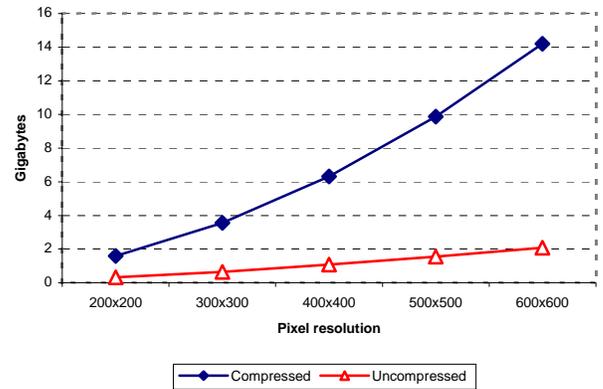

Figure 7. The total size of the light field databases constructed, with and without compression.

Using 32 processors, the time needed to generate the light field database, including the compression step, ranges from 2 to 4.5 hours as the image resolution increases from 200×200 to 600×600. Most of the time spent is on disk I/O operations. After compression, the size of each view set varies from view to view, but on average, at resolutions 200×200 to 600×600 ranges from 1.2 MB to about 7.8 MB.

4.2 Streaming Experiments

In all of our experiments, the client and client agent were run on different machines (both PCs with 2.4 GHz Pentium 4 processors) within the LAN of our department. The word "LAN" in our following discussion always refers to the LAN where the client and client agent are colocated. We ran tests for three cases as follows:
1. LFD stored in LAN, driven by client agent pre-fetch.

2. LFD stored remotely in California and streamed by pre-fetching initiated by client agent.

3. LFD stored remotely in California, aggressively pre-staged on a local depot in LAN and pre-fetched by client agent from the LAN depot.

Case 1 is really local area streaming, since no WAN involvement is present. This is the ideal case of remote visualization, since the WAN appears to be performing in the same way as a LAN. Case 2 is the classical example of prefetching over WAN, often showing excessively long delays due to the uncontrollable nature of Internet. Case 3 takes advantage of the scalability and flexibility offered by LoN infrastructure. Our following results will show that, aided by logistical networking, remote visualization across WAN can be run with almost the same user experience as Case 1. In all three cases, the same prefetch policy, described in Section 3.5, is used by the client agent.

Our tests in all three cases at different image resolutions of LFD's are run with the same sequence of user input, i.e. movement of cursor. We enforce this by using a standard list of cursor movements to orchestrate each test. The user movements are timed as close to identical as is possible using human generated cursor movement. We acknowledge that a more comprehensive study is necessary to understand the impact of user movement patterns.
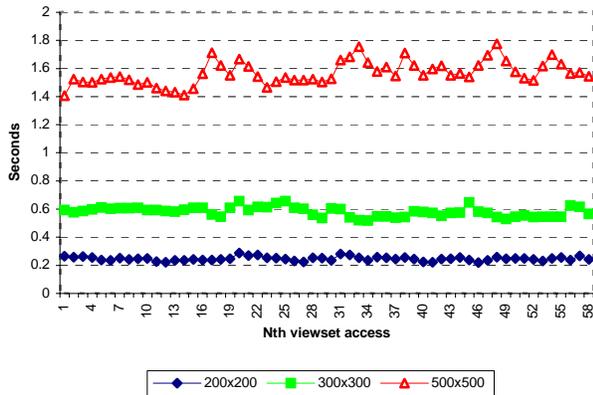

Figure 8. The time to uncompress received view sets during the process of orchestrated user interactions, for image resolutions of LFD ranging from 200 by 200, 300 by 300 and to 500 by 500.

The view sets remain losslessly compressed until received by the client. The time to uncompress a view set varies during the sequence of user interaction. We recorded such overhead in Figure 8. The time to uncompress the view set is counted as part of the latency observed by the client in receiving a view set it requested. Our test shows the decompression of resolutions below 400 to be sub-second. Such resolution corresponds to lightweight devices such as PDAs. This implies that for those low-end devices it is sufficiently fast for a client to request a new view set whenever it needs to, without any local caching on the

client at all. When it comes to higher image resolutions, however, the decompression time is not negligible in an interactive application any more. Accordingly, on devices such as laptops and personal workstations, some level of local caching on the client is also necessary. Given today's computing technology, such system requirements are very reasonable. Alternatively, a more efficient compression scheme can be used.

After a view set is decompressed, it can be rendered at above 30 frames per second on the client console due to the simplistic nature of light field rendering algorithms. Such frame rates remain above 30 frames per second even at large image resolutions of 500×500.

When prefetching and client agent caching are enabled, latencies to obtain a new view set from a server depot could be hidden from the client, provided that the user movement is sufficiently slow. We refer to such sufficiently slow rate of user movement as Quality Guaranteed Rate (QGR). The QGR of case 2, direct streaming and prefetching across WAN, is significantly slower than the QGR's in case 1 and 3. Given our current results, the latter rates can be supported very well on devices like PDAs, allowing the user to navigate rather freely and quickly across the WAN. But this is not true with the large pixel count of user consoles using viewpoint sizes above 500×500.

4.3 Experimental Results

In our series of experiments in cases 1, 2 and 3, the dataset is initially stored in its original depot, and the cache for prefetch on client agent (all cases) and the LAN depot (case 3) are empty. As soon as visualization of a dataset begins, aggressive prestaging to the LAN depot is initiated, and continues uninterrupted until the entire dataset has been localized. Prestaging of individual view sets is ordered by distance from the current position of the cursor, and this order is updated dynamically as the cursor moves. Each view set requested by the client is accessed locally if present in the client agent cache (a *hit*); otherwise *from the LAN depot* if it has been prestaged; or if neither of these apply, it is accessed directly *from the WAN*.

While we discuss the data as if it were stored on a single WAN depot and prestaged to a single LAN depot for simplicity, the actual configuration is more complex. The view sets are in fact striped across three depots in California, and when prestaged, they are striped across four depots attached to the client agent by a 1Gb/s LAN. The local area network connecting the depots to the client agent had no other traffic during these experiments.

In all cases, the time to access a view set across the wide area network is fairly constant throughout the experiment. As we will explain below, in all cases using a LAN depot, there is an initial phase during which accesses are made and latency (as measured at the client) is comparable to that of accessing data directly from the WAN. Any improvement during this phase is quantitative

rather than qualitative in nature, and we do not attempt to characterize it here. After that phase, there are no accesses to the WAN and the latency due to accesses to the LAN depot is comparable to that of a hit (as measured at the client). Thus, during this second phase, the latency (as measured at the client) is comparable to that of completely local browsing (hit rate 100%).

Although we ran multiple experiments under the same sequence of user inputs without identical time intervals, the qualitative characterizations we obtained appear quite stable over multiple experiments, and we have a high degree of confidence in their validity. Unfortunately, due to difficulty of carrying out experiments under identical situations, we must delay the presentation of more definitive experimental results to future work.

Figures 9-12 illustrate the results of the nine experiments. In each case, cursor movements by the user generate a sequence of 58 view set requests. All figures compare the performance of an experiment with data accessed in cases 1 through 3.
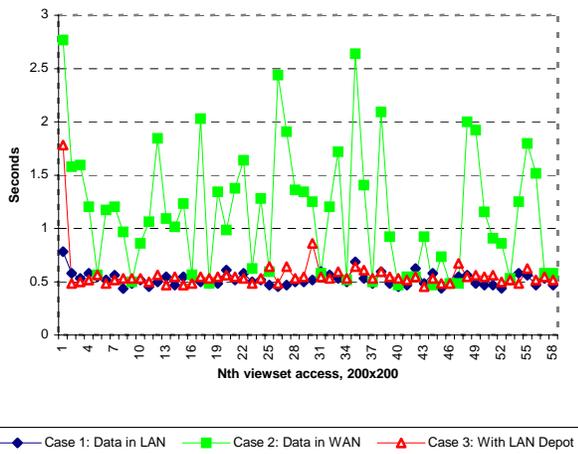

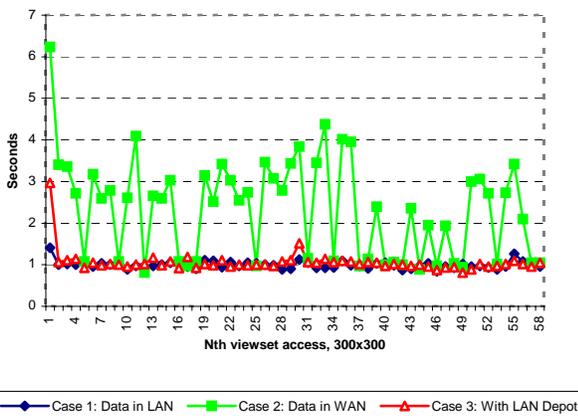Figure 9. Latency as measured at the client for a resolution of 200 by 200

Figure 10. Latency as measured at the client for a resolution of 300 by 300.
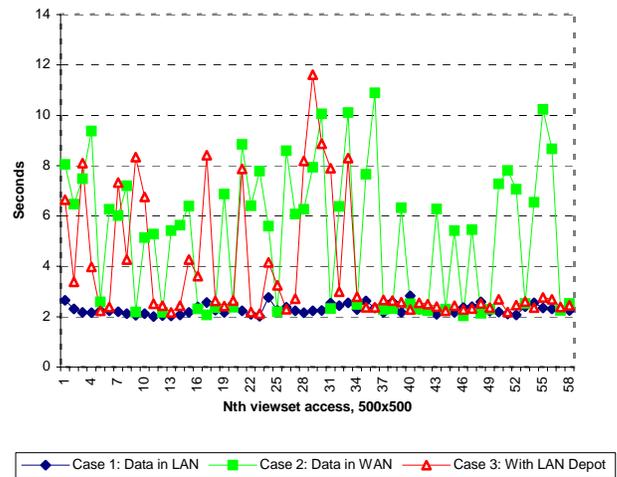

Figure 11. Latency as measured at the client for a resolution of 500 by 500

At resolutions of 200×200 and 300×300, the initial phase lasts for only a single access (Figures 9 and 10). Thus, the user experience when browsing remote data using a LAN depot is quite comparable to browsing data that is stored locally at the client agent. At a resolution of 500×500, the initial phase lasts 33 accesses (Figure 11).

If we examine the results more closely, we can gain insight into the overall performance of the LAN depot. We are particularly interested in its performance during the initial phase at a resolution of 500×500. If this phase can be shortened, or performance during this phase can be improved, then perhaps the striking effectiveness of the LAN depot at low resolutions may be extended to higher resolutions. The following discussion is presented only as observations about our initial data; further experimentation is required in order to make a meaningful analysis.

At lower resolutions, and during the second phase of the 500×500 case, the overall latency as measured at the client is 0.5-2.0 sec (Figures 9-11). This includes latency due to communication, decompression, and any other processing. The communication latency due to data access is on the order of .0001 sec for hits, on the order of .01-.1 sec for access to the LAN depot (Figure 12). This explains why latency of access to data from the LAN depot is indistinguishable by the user from that of a hit or that of access to a database stored in the LAN (case 1). However, the latency due to WAN access is on the order of 1 sec.

During the initial phase, at a resolution of 500×500, the rate of accesses to the WAN is 28% in case 3, compared to 69% in Case 2. However, there is another significant effect that reduces the impact of this improvement: the latency of access to the LAN depot is significantly increased, to the point where it is comparable to that of access to the WAN. Because of this effect, the hit rate is a better, if somewhat conservative, indicator of performance:

33% when using a LAN depot, as compared to 28% without one. This indicates some improvement of case 3 over case 2 in the initial phase, but perhaps not a significant one.

We present no further quantification of the communication latency of access to the LAN depot during the initial phase because there are too few data points to be statistically significant. However, some possible reasons that this latency is so high include these:

- Prefetching of data from the WAN may place a burden on the system resources of the LAN depot that reduces its performance in serving the client agent. Suppressing prefetching while processing a miss may reduce this effect.

- Currently, when a complete view set is not available from the LAN depot, the client agent accesses it directly from the WAN, resulting in redundant transfer of data, placing the burden of WAN access on the client agent, and not making use of any partial transfer already performed to the LAN depot. Optimized routing of all transfers through the LAN depot may reduce this effect.

Clearly, rigorous analysis and optimization of the LAN depot is an area for further investigation. However, this discussion suggests that the excellent performance obtained at lower resolutions may be extended somewhat.

5. Conclusion and Future Work

In this paper we presented a system of remote visualization that addresses typical performance problems stemming from high network latencies and server workloads. First, contrary to previously accepted opinion, we show that although light field databases are large in size, they can, when reorganized using the concept of view set to increase locality, show enough locality and coherence to be exploited for interactive remote visualization. Second, we demonstrated the benefit of the newly developed Logistical Networking infrastructure by applying it to the support of this type of remote visualization. These experiments clearly show the great potential for remote visualization of having, as needed, network caches in the client LAN, multiple sources for data downloading, and higher sustainable network bandwidth provided by the LoN technology.

In particular, LoN infrastructure support allowed us to extend the light field method to the case of remote visualization, where its use is commonly viewed as impracticable. Consequently, remote visualization of datasets in true volumetric forms is supported across the Internet. The clients can make use of consoles with varying levels of functionality, from PDAs to personal workstations. Future work includes the systematic testing of the scalability of our system, both in terms of the number of users and the complexity of visualization process. We will continue to develop remote visualization systems for flow fields and time-varying simulations as well.
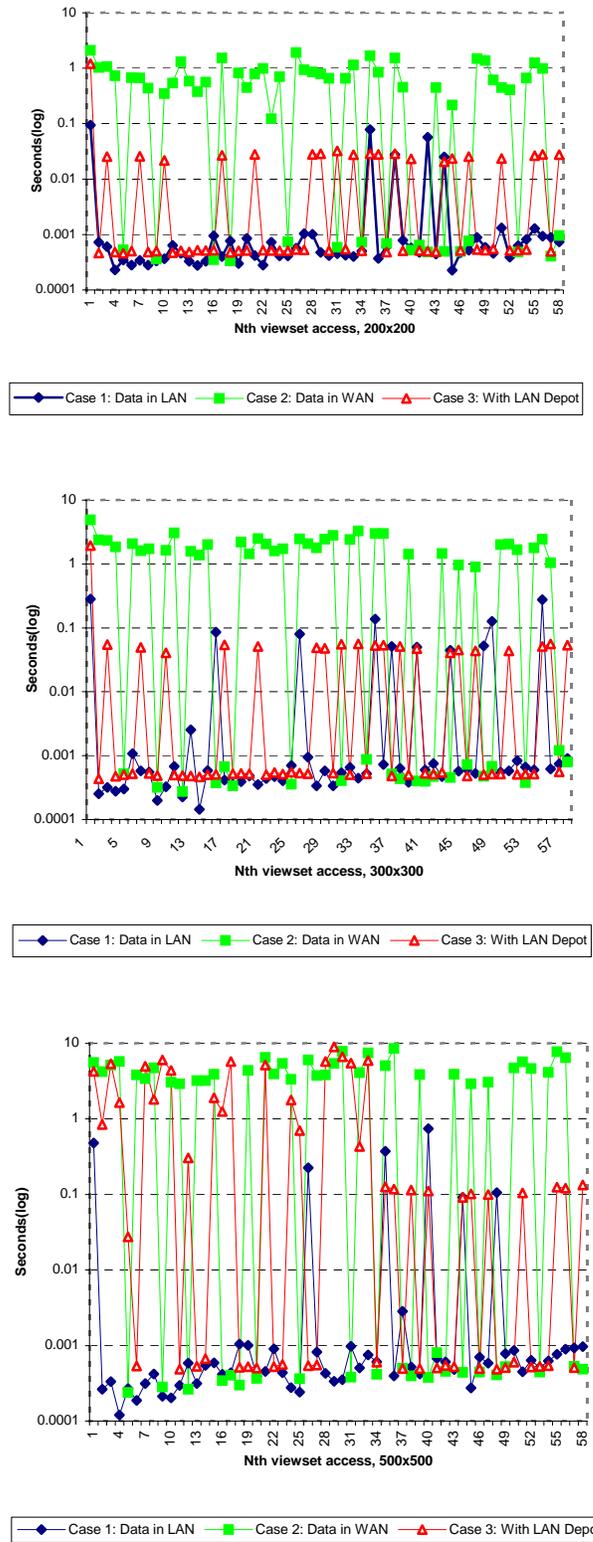

Figure 12. Communication latency due to data access for a resolution of, 200 by 200, 300 by 300 and 500 by 500, respectively as measured at the client agent, shown in log scale.

10

6. References

1. http://www.gzip.org/zlib/.

2. Beck, M., et al., Logistical Networking: Sharing More Than the Wires, in Active Middleware Services, S. Hariri, C. Lee, and C. Raghavendra, Editors. 2000, Kluwer Academic Publishers: Boston.

3. Beck, M., T. Moore, and J.S. Plank, Exposed vs. Encapsulated Approaches to Grid Service Architecture, in Grid Computing -- GRID 2001, C. Lee, Editor. Springer Verlag: Denver, CO. p. 124-132.

4. Beck, M., T. Moore, and J.S. Plank. An End-to-end Approach to Globally Scalable Network Storage. in ACM Sigcomm 2002. Pittsburgh, PA: Association of Computing Machinery.

5. Bethel, W., Visualization Dot Com. IEEE Computer Graphics and Applications, 2000. 20(3): p. 17-20.

6. Bethel, W. and J. Shalf, Cactus and Visapult: An Ultra-High Performance Grid-Distributed Visualization Architecture Using Connectionless Protocols. IEEE Computer Graphics and Applications, 2003.

7. Bethel, W., et al., Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization (LBNL-45365). in Proceedings of SC2000. 2000: Dallas, TX.

8. Gortler, S., et al. The Lumigraph. in Proc. SIGGRAPH '96.

9. Levoy, M. and P. Hanrahan. Light field rendering. in Proc. SIGGRAPH '96. 1996. New Orleans, LA.

10. Li, P., et al., Prefix -- A Parallel Splatting Volume Rendering System for Distributed Visualization, in Proceedings of Parallel Rendering Symposium. 1997. p. 7-14.

11. Liu, Z., A. Finkelstein, and K. Li, Improving Progressive View-Dependent Isosurface Propagation. Computers & Graphics, 2002. 26(2).

12. Luke, E. and C. Hansen. Semotus Visum: A Flexible Remote Visualization Framework",. in Proc. of IEEE Visualization Conference. 2002. Boston, MA.

13. Mueller, K., et al., IBR Assisted Volume Rendering. IEEE Visualization, 1999(Late Breaking Hot Topics): p. 5-8.

14. Plank, J., et al., Algorithms for High Performance, Wide-Area, Distributed File Downloads. 2002, Technical Report CS-02-485, University of Tennessee Dept. of Computer Science.

15. Plank, J.S., et al., Managing Data Storage in the Network. IEEE Internet Computing, 2001. 5(5): p. 50-58.

16. Yang, L. and R. Crawfis. Rail-Track Viewer, an Image Based Virtual Walkthrough System. in Eurographics Workshop on Virtual Environment. 2002. Barcelona, Spain.

17. Zhang, X., C. Bajaj, and W. Blanke. Scalable Isosurface Visualization of Massive Datasets on COTS-Cluster. in Proc. of IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics. 2001. San Diego, CA.