A Case for Peer-to-Peer 3D Streaming

Shun-Yun Hu, Shao-Chen Chang, Wei-Lun Sung, and Jehn-Ruey Jiang

***Abstract*— Interactive 3D contents on the Internet have yet become popular due to their large data volume and the limited network bandwidth. Progressive content transmission, or *3D streaming*, thus is necessary for real-time content interactions and manipulations. However, the heavy data and processing requirements of 3D streaming challenge the scalability of current client-server-based delivery methods. We propose the use of *peer-to-peer* (P2P) networks to make 3D streaming more scalable and affordable, so that interactive 3D contents may see wider adoptions.**

We also describe a conceptual model and a design framework, called *FLoD*, for P2P-based 3D streaming that supports multi-user *networked virtual environments* (NVEs) such as *Massively Multiplayer Online Games* (MMOGs). FLoD allows clients to obtain relevant 3D data from other clients while minimizing server resource usage. Evaluation of FLoD through simulations shows that P2P-based 3D streaming can be fundamentally more scalable than existing client-server approaches.

***Index Terms*— 3D streaming, peer-to-peer (P2P), networked virtual environment (NVE), scalability, overlay networks, visibility determination**

I. Introduction

3D STREAMING refers to the continuous and real-time delivery of 3D contents (such as meshes, textures, animations, and scene graphs) over network connections to allow user interactions without a full download. Similar to audio or video *media streaming* [1], 3D contents need to be fragmented into pieces on a server, before they can be transmitted, reconstructed, and displayed at the client side. Unlike media streaming, as each user often has a different visibility or interest area, the transmission sequence in 3D streaming varies from user to user and may require individualized visibility calculations [2] (i.e. akin to everyone watches a video with unique editings).

There are mainly four types of 3D streaming: object streaming, scene streaming, visualization streaming, and image-based streaming. In this paper we will focus on 3D *scene streaming* for a potentially large *networked virtual environment* (NVE) [3] with many 3D objects, where users can navigate and possibly communicate with one another in real-time. Existing 3D streaming all adopts the client-server architecture as the main delivery model. While this may be fine for a few users, the data and processing-intensive nature of 3D streaming demands prohibitively vast amount of server-side bandwidth and CPU resources when serving a large audience. 3D applications with a large volume of contents (e.g. video games) thus usually require the contents be obtained through pre-installations via CDs or full downloads. However, current application trends point to a need for real-time 3D streaming on possibly a massive scale:

- *Google Earth* is an application that has provided real-time viewing of detailed satellite images of the globe to over a million Internet users. Extending the image-based Earth into a fully 3D one may only be a matter of time, as indicated by initiatives such as *X3D Earth* [4]. However, pre-installing a full 3D Earth for all users is unpractical.
- *Massively Multiplayer Online Games* (MMOGs) [5] are Internet games where up to hundreds of thousands of users interact simultaneously in massive 3D worlds. The most popular titles often have millions of global users. MMOGs' growth and popularity, and their extensions to areas beyond entertainment, may promote a need for content delivery easier than the current CD installations.

Scalable and efficient 3D streaming thus may be an important enabler for diverse forms of new Internet applications. We propose the use of *peer-to-peer* (P2P) networks to improve the scalability and affordability of 3D scene streaming, based on the observation that users navigating through a 3D scene may own similar contents due to overlapped visibility. Users thus might obtain relevant contents from one another. Although there has been significant work for P2P-based media streaming in recent years, it is not directly applicable to 3D streaming as the nature of 3D contents differs from other media types. Novel understandings to the fundamental problems involved and the design of new streaming techniques thus are necessary.

The contributions of this paper are two-fold. First, we formulate a conceptual model for realizing real-time 3D scene streaming on P2P networks by identifying the basic 3D streaming issues as the *fragmentation* of objects and the *prioritization* of transmission order. Additional issues of *scene partition* and *peer and piece selection* are introduced when 3D streaming is adapted to P2P networks. Second, we present the design and evaluation of *FLoD (Flowing Level-of-Details)*, a scalable P2P framework that supports 3D scene streaming for applications such as *X3D Earth* or MMOGs. By identifying and separating the graphics and the networking aspects of the system, FLoD allows researchers from both communities to tackle and improve each aspect independently.

The rest of the paper is organized as follows. Section II provides background on the major themes of this paper. In Section III, we present our model for P2P-based 3D scene streaming. We describe the design of FLoD in Section IV, and its evaluation in Section V. The paper concludes with a summary and descriptions of future work in Section VI.

A preliminary version of this paper appeared in the *Proc. of the 11th Intl. Conf. on 3D Web Technology (Web3D 2006)* under the title: "A Case for 3D Streaming on Peer-to-Peer Networks". This work was supported by NSC of Taiwan, R.O.C. under grant no. NSC 95-2221-E-008-048-MY3.

The authors are with the Department of Computer Science and Information Engineering, National Central University, Taiwan, R.O.C. (e-mail: syhu@yahoo.com; cavour@acnlab.csie.ncu.edu.tw; kergy@acnlab.csie.ncu.edu.tw; jrjiang@csie.ncu.edu.tw. Mailing address: 300 Jhongda Rd., Jhongli City, Taoyuan County 320, Taiwan (R.O.C.) Phone: +886-3-4227151#35312 Fax: +886-3-4222681)

II. BACKGROUND

We first provide some background related to the main themes of this paper (i.e. 3D streaming and P2P networks). Specifically, we will describe work related to P2P media streaming and P2P networked virtual environments.

A. 3D Streaming

In general, the main goal of 3D streaming is to provide 3D contents in real-time for users over network links, such that the interactivity and visual qualities of the contents may match as closely as if the they were stored locally [6]. The resource bottleneck is often assumed to be the bandwidth and not rendering or processing power [2]. To achieve this, simplification and progressive transmission are two dominant strategies [7]. Existing 3D streaming techniques may be categorized into four main types: object, scene, visualization, and image-based streaming, which we describe as follows:

1) Object streaming: Hoppe introduced the concept of *progressive meshes* (PM) [8], which store an arbitrary triangular mesh as an appearance-preserving but much coarser *base mesh* and a number of refinement pieces. A remote user may view or interact immediately with the object once the base mesh is downloaded. Streaming additional pieces incrementally refines the base mesh and restores the original mesh exactly. Geometrical meshes thus can be streamed from servers to clients, making interactions with 3D data possible without a complete download. Progressive meshes were the basis that sparked much subsequent research (e.g. *view-dependent* PM [9], compressed PM [10], over lossy transmission links [11], over wireless channels [12], and QoS-related streaming [13]). For high resolution models, streaming of *QSplat* (a non-polygonal point representation) was investigated in [7]. Object streaming has also been studied in the context of *geometry image* [14], specific file formats (e.g. X3D [15] and MPEG-4's BIFS [16]), and data types besides polygonal meshes, such as textures [17], animations [18], and scene graphs [19].

2) Scene streaming: Object streaming extends naturally to *scene streaming*, where a collection of objects are placed arbitrarily in space and streamed with their placement information to clients according to user visibility or interests. Scene streaming usually aims to provide a *remote walkthrough* (i.e. navigation) or multi-user NVE experience. As many more objects may exist than what the user can see at a given time, scene streaming generally has two stages: *object determination* and *object transmission*. For the first stage, the server employs visibility determination techniques to cull away irrelevant objects, and uses visual quality estimates to assign transmission priorities. For the second stage, data reduction techniques such as progressive representations and compressions are used for sending the object pieces (i.e. similar to single object streaming). Scene streaming also benefits from the reuse of cached contents, so that objects need not be sent again if they are re-visited later in the walkthrough [2]. Many techniques useful for scene streaming were first described by Schmalstieg and Gervautz [20], where each user's visibility is limited to a circular *area of interest* (AOI). A server determines and transmits the set of visible objects at different *level of details* (LODs) to clients. Clients also *prefetch* objects to mask the download latency from users. Their subsequent work replaced the use of discrete LODs with continuous (*smooth*) LODs and named the process *remote rendering* [21]. Teler and Lischinski proposed the use of pre-rendered image-based *impostors* as the lowest LOD of 3D objects to allow faster initial visualizations [2]. The server also uses an *online optimization algorithm* to choose suitable contents that may provide the best visual quality under a limited bandwidth budget. Immediate and practical navigation thus is possible even with a 2000 bytes/s bandwidth. *Cyberwalk* [22] adopts progressive meshes to avoid the data redundancy from sending multiple LODs. It also focuses on caching and prefetching techniques to enhance visual perceptions and reduce the *response time* to obtain objects. Deb and Narayanan propose a *geometry streaming system* that focuses on maintaining interactive frame-rate by adaptive data selection according to the client capabilities and network conditions [6]. Some social MMOG systems utilize scene streaming to support dynamic contents (e.g. ActiveWorlds, There.com [23], and Second Life [24]), but few public information is available on their mechanisms.

3) Visualization streaming: Certain scientific computing generates vast amounts of data that requires visualization in 3D spaces for analysis or comprehension. Streaming for these 3D data differs from object-based streaming in that the data volume is usually much larger, and may involve time-dependent model deformations that require complete refreshes (i.e. re-download) of the dataset. Accuracy of model representations is also given priority over visual aesthetics. Olbrich and Pralle pioneered a series of such systems for scientific visualizations based on VRML and a customized *DocShow-VR* (DVR) format [25]. Another system, *ViSTA* [26], focuses on interactive visualizations of *computational fluid dynamics* (CFD), where a server-cluster is used for the parallel generations and post-processing of raw data to allow view-dependent visualizations. Such streaming systems call for an environment of high performance networks and graphics servers to pipeline the processing of large data volumes. They are thus not suitable on the Internet where the bandwidth is often limited and graphics servers may not be available.

4) Image-based streaming: In *image-based streaming*, 3D contents are stored at the server only. Clients instead receive 2D rendered images generated in real-time by the server [27]. This approach consumes only constant bandwidth, and is suitable, maybe even necessary, when the client has only thin functionalities (i.e. low processing power with no 3D acceleration capability, such as hand-held devices). However, the severe processing requirements on the server may cause poor scalability and interactivity.

B. Peer-to-Peer Networks

P2P networks have gained publicity and popularity through file-sharing applications in recent years, yet its central concept – a distributed network where participants contribute resources to support collective tasks – is applicable to a wide range of uses (e.g. *distributed hash table* [28], *voice-over-IP* [29], and *web cache* [30], etc.).

Existing file-sharing mechanism such as *BitTorrent* [31] has demonstrated the feasibility to efficiently distribute large files by dividing them into small pieces and utilizing the uplink bandwidth of other peers. Although typical bandwidth on today's end-user networks is asymmetric, where the "uplink" is less than the "downlink" bandwidth, by downloading from multiple peers in parallel, good performance can still be achieved. For example, average download between 240kbps to 500kbps were reported in measurement studies [32], [33].

Media streaming on P2P networks has been studied extensively in recent years and mainly falls into two categories: synchronous (i.e. live) streaming [1] and asynchronous (i.e. on-demand) streaming [34]. Although *IP multicast* is ideal for disseminating identical data streams to many receivers, its lack of deployment has motivated the designs of *application-layer multicast* (or *end system multicast* [35]), where multicast trees are built with the clients as nodes, and the server as root. Issues such as minimizing the tree depth, balancing each node's link-degree, and maintaining the tree structure after node departures thus are the main issues. Although similarities exist between 3D and media streaming (e.g. data stream is usable before a full download, sequential data block transfer, and the applicability of prefetching), they differ in one fundamental aspect: the contents and transmission order of 3D streaming are not static but based on the results of visibility calculations. As each user may have different visibilities, individual data streams are mostly unique unless users are in close proximities with each other [2]. In other words, *3D streaming works as if every individual user is watching a unique home movie, recorded by different cameras within the same scene*. Due to this major difference, existing P2P media streaming techniques may not be straightforwardly applied to 3D streaming.

The study of multiuser networked virtual environments seeks to allow people at different places to interact in the same virtual world as seamlessly as possible [3]. NVE's network model has evolved from the least scalable *point-to-point* (i.e. all participating users exchange messages directly), to the later *client-server* (i.e. a centralized server receives, processes, and relays messages for all users) and today's *server-cluster* models. However, as server-based architectures likely will not scale user size to the next order of magnitude, some P2P solutions have recently been proposed [5], [36], [37]. As each user only has limited bandwidth, the premise of P2P-based NVE (P2P-NVE) is that by connecting with only a few nearby users, each user node can obtain the relevant messages within its *area of interest* (AOI). The central challenge in P2P-NVE systems thus is a *neighbor discovery problem*: how to allow every user node to discover the neighbors within its AOI, known as *AOI neighbors*, correctly and efficiently as they move around. Aside from some variations in the correctness and efficiency to maintain the P2P connectivity, most current P2P-NVE proposals can provide the following function without server involvements: given a user node's virtual coordinate and AOI in the NVE (usually specified as a circle centered at the user), return the information of its AOI neighbors such as the neighbors' coordinates and IP addresses. This function will be relevant to our design for FLoD, as discussed later in Section IV.

III. P2P-BASED 3D SCENE STREAMING

In this section, we try to formulate a conceptual model for understanding 3D scene streaming in a P2P context.

A. System Model and Assumptions

We consider a *remote walkthrough* [2], [22] scenario where 3D objects of various sizes and shapes are placed in a large scene with specific positions and orientations. Objects are defined by polygonal meshes and their associated data, such as textures, animations, and light maps, etc.. Each user navigates the scene through a client program (the terms *user*, *node*, *client*, and *peer* will be used interchangeably from now on). As there are potentially many objects, it is neither feasible nor necessary to see and interact with all of them at once. Each user's visibility and interaction thus is limited to a circular AOI centered at the user's current location. For simplicity, we assume that all objects are static in both their positions and contents. In this basic model, we also do not consider displaying 3D representations of other users (i.e. each user can only see static objects, but not each other).

For a given 3D object, we assume that its mesh and other data can be fragmented into a *base piece* and many *refinement pieces* (Fig. 1). The specific fragmentation is beyond our scope, but whichever the mechanism, we assume that the user is provided with a minimal working set of objects once the base pieces are obtained, such that the scene can be rendered and navigation may start. Progressive meshes [8] and related techniques such as geometry image [14] may be used for mesh fragmentation, while progressive encodings of GIF, JPEG, or PNG, may be used for texture fragmentation [17].

All 3D contents are initially stored at a server, and clients obtain them through a streaming process from either the server or other clients. Rendering and navigation may begin as soon as base pieces of a few objects within the AOI are obtained.

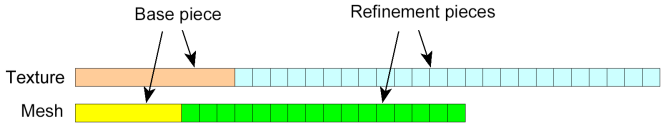


Fig. 1. 3D content fragmentation.

B. Requirements

From the user's perspective, the main concern for 3D streaming is its *visual quality*, which is captured by concepts such as *walkthrough quality* [2] or *visual perception* [22]. However, as visual quality can be a subjective judgement, a more definable concept may be the *streaming quality* in terms of "*how much*" and "*how fast*" a client obtains data. For the former, one measure is the ratio between the data currently owned and those necessary to render a view at an instant, which we will call *fill ratio*. A ratio of 1 indicates the best visual quality, as the rendered image would be the same as if all contents are locally stored. As for the latter, we may use the following two measures: *base latency*, the time to obtain the base piece of an object, and *completion latency*,

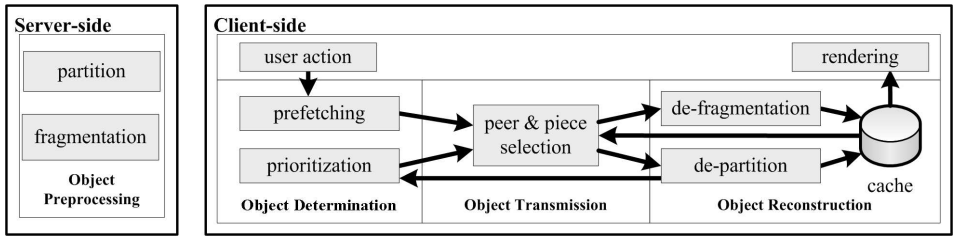


Fig. 2. A conceptual model for P2P-based 3D scene streaming.

the time to download the complete data of an object. Note that the two terms are similar to *latency time* and *response time* in [22]. Base latency indicates the delay for a user to see a basic view of an object, while completion latency indicates the delay of being able to fully inspect or manipulate an object. For clients, the goal of 3D streaming thus is to optimize the streaming quality by maximizing the fill ratio for every view and minimizing the base and completion latency.

From the server's perspective, the main concern is to improve the system's *scalability* by distributing processing and transmission loads to clients as much as possible. For transmissions, it is preferable if most contents are delivered by clients. This can be measured by the amount of server-side bandwidth usage. For processing, it is desirable to minimize the server's role in calculating user visibility and deciding the transmission strategy. Ideally, if these calculations are delegated to clients, then server-side processing can be conserved for answering data requests only. For servers, the goal of 3D streaming thus is to minimize their CPU and bandwidth usage.

C. Challenges

To meet the above requirements by utilizing client resources, we identify two new issues to address:

Distributed visibility determination Preferably, visibility determination should be done without server involvement or global knowledge of the scene. However, as only the server initially possesses complete knowledge of object placements (i.e. the *scene descriptions*, which are required for visibility calculations), we need ways to partition and distribute scene descriptions to clients so that visibility determination can be done in a distributed manner efficiently.

Peer and piece selection To optimize the visual (streaming) quality for a given bandwidth budget, clients should perform *peer selection* to contact the proper peers and *piece selection* to request the proper data pieces for object reconstructions. As there may be multiple relevant data sources, factors such as resource capacity, content availability and network conditions need to be considered together. Additionally, as 3D streaming is view-dependent [9] and that some data pieces may be applied in arbitrary order during object reconstructions. 3D streaming requires only a *roughly sequential* transfer order, which should be considered during piece selection to ensure that piece dependencies are satisfied. On the other hand, where dependencies do not exist, concurrent download may be exploited to accelerate data retrievals.

D. Conceptual Model

Given the above requirements and challenges, we now summarize the main tasks of 3D scene streaming as follows:

Partition: The task of dividing the entire scene into blocks or cells so that global knowledge of all object placements is not required for visibility determination. Scene partition is essential if visibility calculations were to become decentralized.

Fragmentation: The task of dividing a 3D object into pieces so that it may be transmitted over the network and reconstructed back progressively by a client. Progressive meshes or textures are all examples of fragmentation techniques.

Prefetching: The task of predicting data usage ahead of time and generating objects or scenes requests so that latency due to transmissions is masked from users. Predications of user movements or behaviors are often employed for this task [22].

Prioritization: The task of performing visibility determination to generate the ordering for a client to obtain object pieces within a scene. The goal is to produce the best streaming quality with considerations of factors such as object distance, line-of-sight [2], [22], or the requesting client's bandwidth [6].

Selection: The task of determining the proper peers to connect and pieces to obtain based on considerations of peer capacity, content availability and network conditions, in order to fulfill data requests from prefetching and prioritization efficiently.

Fig. 2 shows a coupling of the above tasks into a conceptual model for P2P-based 3D scene streaming. *User action* and *rendering* are the only steps if contents are locally available. *Object preprocessing, determination, transmission,* and *reconstruction* are additional stages in 3D streaming. For client-server-based 3D streaming, only *fragmentation*, *prefetching*, and *prioritization* are considered. *Partition* of the scene and the *selection* of peers and pieces are new issues introduced in P2P-based 3D streaming. Table I shows a comparison between client-server and P2P-based 3D streaming.

TABLE I

TASK COMPARISONS BETWEEN CLIENT-SERVER AND P2P 3D STREAMING.

Processing stage \ Architecture		Client-server	Peer-to-Peer
(offline)	partition	——	Server
Pre-processing	fragmentation	Server	Server
(online)	prefetching	Client	Client
Navigation	de-partition	——	Client
	prioritization	Server/Client	Client
	selection	——	Client
	de-fragmentation	Client	Client

IV. Design of FLoD

In this section, we describe our design of a P2P-based 3D streaming framework that fulfills the requirements in the last section. An overview is given first, followed by the procedures.

A. Overview

FLoD's main design rationale is that as a node often has overlapped visibility with its AOI neighbors, it is likely that the neighbors already possess some relevant 3D contents. By requesting data from the neighbors first, the server can be relieved from serving the same contents repetitively. We assume the existence of a P2P-NVE overlay such as VON [37], which returns the information of AOI neighbors given a node's position and AOI-radius. The information includes the neighbors' IDs, coordinates, and IP addresses. As a node moves around, it updates the overlay with its new position and gets refreshed information on AOI neighbors.

To distribute scene descriptions to clients efficiently, the NVE is partitioned into fixed-size square *cells* (similar to [22]), each has a small scene description specifying the objects within. Each 3D object is specified by a *unique ID*, *location point*, *orientation* and *scale* within the scene description. Selecting visible objects in the AOI can thus be done in a fully-distributed manner, as each node can locally determine the cells covered by its AOI (Fig. 3). When entering a new area, a client first prepares a *scenes request* to obtain scene descriptions from its AOI neighbors or the server. A *pieces request* is then created for the visible objects. Piece dependency is also specified in the request to ensure that data retrieval follows the ordering of object reconstructions. Views are rendered progressively as objects are streamed from either the neighbors or the server (which acts as the final fallback if peers cannot respond due to data or bandwidth unavailability).

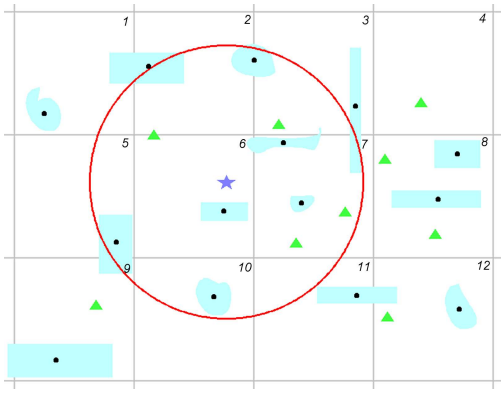


Fig. 3. Schematic of a NVE divided into *cells*. Big circle is the AOI of the star node while triangles are other *user nodes*. Various shapes are 3D objects, with their *location points* as dots. Note that cell IDs can be calculated given the star node's location coordinates, the world dimensions and cell size.

As FLoD seeks to be flexible and extensible in accommodating evolving policies and techniques, we separate the main client-side tasks into a *graphics layer* and a *networking layer* (Fig. 4). The graphics layer performs *object determination* (i.e. prefetching and prioritization) and *object reconstruction* (i.e. de-partition and de-fragmentation), while the networking layer is responsible for *object transmission* (i.e. peer and piece selection). Prefetching and caching are not yet considered in depth, but are included for the sake of completeness. The *application* sits on top of FLoD and performs the traditional 3D application tasks of taking *user actions* and *rendering*.

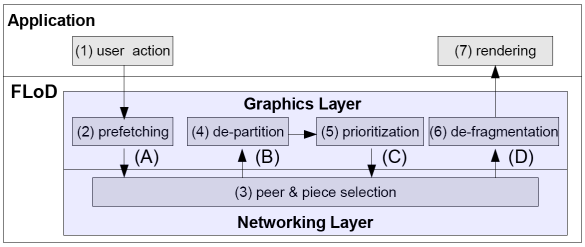


Fig. 4. FLoD's client-side task flow and layers. Data flows: (A) scenes request (B) scene descriptions (C) pieces request (D) data pieces. Numerical labels for the tasks are described in the FLoD *Procedures*.

B. Procedures

We now describe FLoD's main procedures in more details. Tasks performed in each procedure are specified after the procedure names according to the task numbers in Fig. 4:

Login: The joining node enters the P2P network by specifying a join location and AOI-radius to the P2P-NVE *overlay*, which returns an initial list of AOI neighbors. The NVE's dimensions and cell size are also obtained from a *gateway server*. *Obtain Scene Descriptions* procedure is then called.

Obtain Scene Descriptions (2, 4): The requesting node determines the cells that its AOI covers, and uses the *Request for Data* procedure to get the cells' *scene descriptions* by passing a *scenes request* made of cell IDs. Once the *scene descriptions* are obtained and analyzed, the node requests for 3D objects with the *Obtain Objects* procedure.

Obtain Objects (5, 6, 7): Visibility determination produces a prioritized *pieces request*, consisting of (*object ID*, *piece ID*, *depended-piece ID*) tuples, for any missing visible data. Pieces are obtained according to their priorities and dependencies via the *Request for Data* procedure, and stored to a cache once downloaded. A view is rendered from the cache according to objects' specified locations, orientations, and scales in the *scene descriptions*.

Request for Data (3): If the local cache does not contain the desired data, requests are sent to the *data source nodes* (composed of current AOI neighbors and the *gateway server*), according to certain *peer selection policy*. The actual data exchanges are governed by certain *piece selection policy*. As the *gateway server* is part of the pool, requests will eventually go to the server if the peers cannot fulfill them.

Move (1): A node moves by sending a position update to the *overlay*. If new neighbors are discovered, they will become part of the *data source nodes*. If the node has entered any new cells which it does not have the *scene descriptions*, *Obtain Scene Descriptions* is invoked.

Logout: A node may simply disconnect from the P2P network. As the system is fairly distributed, failure or departure of any single user node will not affect the system's operation. Other nodes will learn about the departing node through updated neighbor list provided by the *overlay*.

V. EVALUATION

To evaluate FLoD's design, we perform simulations using a discrete-time simulator. In this section, we present our simulation policies, metrics, setup, results, and discussions.

A. FLoD Policies

As FLoD is a flexible framework where different policies may be adopted for various tasks, we first describe the policies used in this initial evaluation of FLoD:

Prefetching and caching: We do not use any prefetching in the simulation as it is not our main focus. The cache size is set to 15MB where the farthest object is replaced first when the cache size limit is exceeded [22].

Prioritization strategy: The highest priority is given to the base pieces (i.e. piece id = 0). Subsequent pieces are inserted to the *pieces request* by rotating the next piece from each desired object (i.e. we assume that for a given object, each piece depends only on the previous piece). For example, if the *pieces request* consists of *(object_id, piece_id, depended-piece_id)* tuples, for requesting object #1, #2, #3 the list would look like (1,0,-) (2,0,-) (3,0,-) (1,1,0) (2,1,0) (3,1,0) (1,2,1) (2,2,1) (3,2,1), etc..

Peer and piece selection: A node first queries all known AOI neighbors for the availability of the requested data. A request is then sent randomly to a positively responded neighbor. If the data is unavailable, a node will re-query until either 1) it becomes the nearest node to the requested object or 2) it is within 20 units of the requested object, then the request is sent to the server. For now, our piece selection policy is to simply select the next piece in the *pieces request*, as 3D contents need to be obtained more or less sequentially.

B. Simulation Metrics

The purpose of the simulation is to compare the *scalability* and *streaming quality* between a P2P and a client-server approach of 3D streaming, in respect to the following metrics:

Bandwidth usage: One fundamental requirement for scalable systems is that resource usage at each system component (i.e. server or client) is *bounded* without exceeding the component's capacity. Otherwise an overloaded component may fail or degrade its service quality. Bandwidth usage at all nodes and the server, thus are important indicators for system scalability.

Fill ratio: 3D streaming aims to achieve a visual quality matching that of locally stored contents. Here we measure the ratio of data volumes between the client's obtained data and visible data (according to the server's storage), to estimate a client's capability of rendering a view.

Base latency: We define the time between the initial query and the time a base piece becomes available at a client as *base latency*. It serves as an indicator for how soon a user may start meaningful navigation when entering a new scene.

Peer hit ratio: The success of data requests made to other peers may influence both the base latency and the transmission overhead for re-requests. We thus define *peer hit ratio* as the percentage of object pieces that are successfully obtained from those requested. This may indicate the efficiency of the peer selection and caching policy.

TABLE II
SIMULATION PARAMETERS

World dimension (units)	1000x1000
Cell size (units)	100x100
AOI-radius (units)	75
Time-steps	3000
Number of nodes	100 - 1000 (in 100 increment)
Number of objects	500
Node speed (units / step)	1
Client cache size (MB)	15

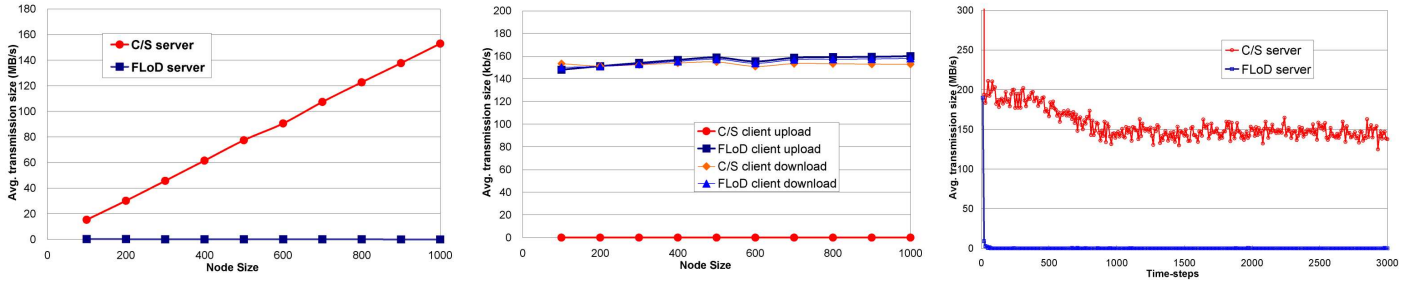
C. Simulation Setup

We choose *Voronoi-based Overlay Network* (VON) as the underlying P2P overlay, as it has demonstrated scalability, consistency, and reliability [37]. Note that FLoD may also use other P2P-NVE overlays, as long as correct and timely information on AOI neighbors are provided. One benefit of VON is that a few nearest neighbors are always maintained even when none is within the AOI. Requests to peers thus are still possible in such cases. As the main purpose of this work is to compare the resource usage patterns between client-server and P2P 3D streaming, we assume that both the server and clients have unlimited bandwidth. Although this assumption does not apply to the current Internet, it does approximate some existing high-speed military simulation networks.

For the simulations, we first randomly place a number of objects on a 2D map partitioned into square cells. For simplicity, we assume that each object has only one set of pieces (i.e. we assume that each piece contains a combination of various types of data such as meshes or textures, enough for object reconstructions). Object sizes range randomly between 100kb and 500kb, with 20% of the size as the base piece, and 50kb for each refinement pieces. We then create a number of nodes, each moves with a constant speed using *random walk* as the movement model [22]. Scene descriptions or data pieces are requested from AOI neighbors as needed. The simulation proceeds in discrete *time-steps*, where a node may either process or send messages to others in each step. Assuming each step is 100ms, our 3000-step simulation is equivalent to running a system for 5 minutes. Statistics are collected after 200 simulation steps when transmissions have stabilized. Specific simulation parameters are shown in Table II.

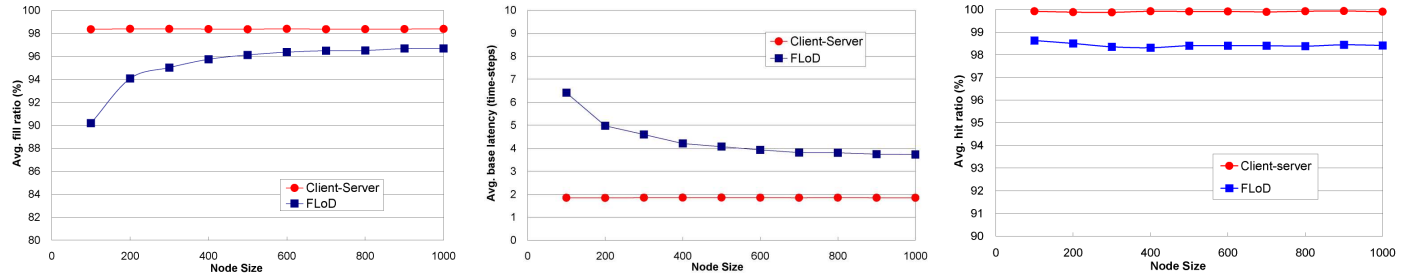
D. Simulation Results

Scalability The amount of per-second server upload size grows linearly for client-server (C/S) but is significantly lower for FLoD at less than 1 MB/s (Fig. 5(a)). For clients, Fig 5(b) shows that the the per-node per-second download size is around 160kb/s for C/S and slightly more for FLoD, but both remains constant regardless of node size. This is due to our assumption of uniform object distributions and constant node speeds, so that the average amount of additional contents to render a client view stays constant. Client uploads in FLoD are much higher than C/S and roughly equals to downloads as the clients are capable to service other clients. Fig. 5(c) shows a time-series of server upload size for 1000 nodes, where the



(a) server transmissions (upload) (b) client transmissions (upload and download) (c) server transmission time-series (1000 nodes)

Fig. 5. Bandwidth usage comparisons (Average transmission size per node per second)



(a) fill ratio (b) base latency (c) peer hit ratio

Fig. 6. Streaming quality comparisons

transmissions stabilize after 1000 steps for C/S, and remains low at less than 1MB/s for FLoD. Note that the initial server transmission is exceptionally large (e.g. 2207MB/s for C/S and 190MB/s for FLoD) as all the initial requests are answered by the server without bandwidth limits. This allows contents be quickly disseminated to all peers in the beginning. The cost to maintain the overlay is about 10-20 KB/s in all cases, which is relatively small compared to content transmissions.

Streaming Quality We use *fill ratio*, *base latency*, and *peer hit ratio* to measure the streaming quality of a system. From Fig. 6(a) we see that fill ratios are quite high (i.e. above 98% for C/S and above 96% for FLoD when node density is high), which implies that high visual quality can be achieved. As the simulation assumes unlimited bandwidth, the base latency in C/S is less than 2 time-steps, as requests can be fulfilled by either the local cache (i.e. latency = 0 step) or the server within one round-trip request (i.e. latency = 2 steps) (Fig. 6(b)). Base latency in FLoD is somewhat higher, but gradually decreases to below 4 time-steps, indicating that roughly 2 queries are attempted before a data request succeeds. The peer hit ratio is 100% for C/S and above 98% for FLoD (Fig. 6(c)). The high ratio is due to the design that FLoD clients request data from other peers only after data availability is confirmed.

E. Discussions

Scalability Simulations have shown that FLoD significantly reduces the server-side bandwidth usage, and bounds the transmissions for clients. This indicates that P2P-based 3D streaming can be *fundamentally* more scalable than client-server approaches as it is possible to prevent either the server or clients to become a resource bottleneck.

Distributed visibility determination By pre-partitioning the NVE into cells and limiting a node's visibility to its AOI, visibility determination thus can be done without server involvements. This helps scalability as the server does not need to calculate visibility for any node in the system.

Peer and piece selection The challenge of finding the peers with relevant content is solved in two stages: 1) A list of AOI neighbors is obtained by using a P2P-NVE overlay. 2) A FLoD client would query its neighbors a few times, before requesting data from the server. This allows some time for a client to locate the desired contents from its peers, reducing the number of server requests. The piece selection is a simple sequential one due to the assumption of linear piece dependency.

Layered framework By separating the main tasks into a graphics and a networking layer, and defining a clear interface between them, each layer may thus be independently improved by experts from both the graphics and networking fields without requiring cross knowledge from other field. For example, peer and piece selections may be improved independently from fragmentation techniques.

Limitations In the current design, each node retrieves data pieces from only their AOI neighbors, which might not be the complete set of qualified nodes, and sufficiently large number of peers must be within the AOI for it to work. Efficiency at matching data requests thus might not be optimal. We also have not investigated caching or prefetching in depth, however, they are essential for any streaming scheme to be effective. We assume piece dependency to be linear, which may be too strict to exploit enough download parallelism. The display of other users also has not been considered, but it can be easily supported as AOI neighbors are already known by each node.

VI. Conclusion

We have formulated a conceptual model for P2P-based 3D scene streaming, and identified the main tasks as the partition of scenes, the fragmentation of objects, the prefetching of potentially visible objects, the prioritization of transmission order, and the selection of peers and pieces for deliveries. We have also presented FLoD, a scalable P2P 3D scene streaming framework where the neighbor discovery mechanism of P2P-NVE overlays is used to discover and maintain relevant peers having shared contents. Distributed visibility determination and peer and piece selection also relieve the server from intensive computations and transmissions. Simulations show that FLoD is fundamentally more scalable than client-server architectures by bounding both the server and the clients' bandwidth usage. An open source implementation of FLoD is available at: http://ascend.sourceforge.net.

There are a number of directions for future work:

- Considerations of bandwidth limitations
- Development of more efficient peer and piece selections with considerations to piece dependency
- Prioritization strategies that differentiate between different types of 3D contents
- Accelerated data retrieval from non-AOI neighbors
- Prefetching and caching schemes to mask transfer latency
- Display of other users within the AOI
- Application of FLoD to practical systems

Real-time 3D contents have yet found a way to most Internet users in spite of years of efforts. While challenges remain in areas such as format standards and the ease of content creations, content streaming may effectively address the delivery problem. 3D streaming on P2P networks thus is an important topic worthy of the attentions of both graphics and networking professionals. By identifying the basic issues, we hope to generate interests in this promising direction to realize more convenient access to 3D contents.

Acknowledgment

We thank Dr. Bing-Yu Chen for suggesting the issue of piece dependency, Huang Ting-Hao, Nien-Hsien Lin, Ye-Zen Chang Chen and Chen-Yu Yeh (Yakko) for valuable discussions. We are also grateful to Dr. Chin-Kun Hu and Dr. Ming-Chya Wu of Laboratory of Statistical and Computational Physics (LSCP) of Academia Sinica, Taiwan for the computing facilities.

References

[1] D. Tran, K. Hua, and T. Do, "A peer-to-peer architecture for media streaming," *IEEE JSAC*, vol. 22, no. 1, pp. 121–133, 2004.

[2] E. Teler and D. Lischinski, "Streaming of complex 3d scenes for remote walkthroughs," *EUROGRAPHICS*, vol. 20, no. 3, 2001.

[3] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*. ACM Press, 1999.

[4] D. Brutzman, "X3d earth project proposal," http://www.web3d.org/news/presentations/X3dEarth_2006July31.pdf.

[5] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *Proc. INFOCOM*, 2004, pp. 96–107.

[6] S. Deb and P. J. Narayanan, "Design of a geometry streaming system," in *Proc. ICVGIP*, 2004, pp. 296–301.

[7] S. Rusinkiewicz and M. Levoy, "Streaming qsplat: A viewer for networked visualization of large, dense models," in *Proc. ACM I3D*, 2001, pp. 63–69.

[8] H. Hoppe, "Progressive meshes," in *Proc. ACM SIGGRAPH*, 1996, pp. 99–108.

[9] J. Kim, S. Lee, and L. Kobbelt, "View-dependent streaming of progressive meshes," in *Proc. SMI'04*, 2004, pp. 209–220.

[10] R. Pajarola and J. Rossignac, "Compressed progressive meshes," *IEEE Trans. Vis. Comput. Graph.*, vol. 6, no. 1, pp. 79–93, 2000.

[11] Z. Chen, B. Bodenheimer, and J. F. Barnes, "Robust transmission of 3d geometry over lossy networks," in *Proc. ACM Web3D*, 2003, pp. 161–ff.

[12] S. Yang and C.-C. J. Kuo, "Robust graphics streaming in walkthrough virtual environments via wireless channels," in *Proc. Globecom*, 2003.

[13] B.-Y. Chen and T. Nishita, "Multiresolution streaming mesh with shape preserving and qos-like controlling," in *Proc. Web3D*, 2002, pp. 35–42.

[14] N.-S. Lin, T.-H. Huang, and B.-Y. Chen, "View-dependent jpeg 2000-based mesh streaming," in *ACM SIGGRAPH 2006 Conference Abstracts and Applications (Posters Program)*, 2006.

[15] E. Fogel, D. Cohen-Or, R. Ironi, and T. Zvi, "A web architecture for progressive delivery of 3d content," in *Proc. Web3D*, 2001, pp. 35–41.

[16] M. Hosseini and N. D. Georganas, "Mpeg-4 bifs streaming of large virtual environments and their animation on the web," in *Proc. ACM Web3D*, 2002, pp. 19–25.

[17] J.-E. Marvie and K. Bouatouch, "Remote rendering of massively textured 3d scenes through progressive texture maps," in *Proc. 3rd IASTED Conf. VIIP*, vol. 2, 2003, pp. 756–761.

[18] T. Hijiri, K. Nishitani, T. Cornish, T. Naka, and S. Asahara, "A spatial hierarchical compression method for 3d streaming animation," in *Proc. ACM VRML*, 2000, pp. 95–101.

[19] J. Sahm, I. Soetebier, and H. Birthelmer, "Efficient representation and streaming of 3d scenes," *Computers & Graphics*, vol. 28, no. 1, pp. 15–24, 2004.

[20] D. Schmalstieg and M. Gervautz, "Demand-driven geometry transmission for distributed virtual environments," *Computer Graphics Forum*, vol. 15, no. 3, pp. 421–433, 1996.

[21] G. Hesina and D. Schmalstieg, "A network architecture for remote rendering," in *Proc. Intl. Wksp. DIS-RT*, 1998, p. 88.

[22] J. Chim, R. W. H. Lau, H. V. Leong, and A. Si, "Cyberwalk: A web-based distributed virtual walkthrough environment," *IEEE Trans. on Multimedia*, vol. 5, no. 4, pp. 503–515, 2003.

[23] B. Brown and M. Bell, "Cscw at play: 'there' as a collaborative virtual environment," in *Proc. ACM CSCW*, 2004, pp. 350–359.

[24] P. Rosedale and C. Ondrejka, "Enabling player-created online worlds with grid computing and streaming," Gamasutra Resource Guide. http://www.gamasutra.com/resource_guide/20030916/rosedale_01.shtml, 2003.

[25] S. Olbrich and H. Pralle, "Virtual reality movies - real-time streaming of 3d objects," *Computer Networks*, vol. 31, no. 21, pp. 2215–2225, 1999.

[26] A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen, and C. Bischof, "Viracocha: An efficient parallelization framework for large-scale cfd post-processing in virtual environments," in *Proc. SC2004*, 2004.

[27] L. Cheng, A. Bhushan, R. Pajarola, and M. E. Zarki, "Real-time 3d graphics streaming using mpeg-4," in *Proc. IEEE/ACM Wksp. on Broadband Wireless Services and Appl.*, 2004.

[28] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. SIGCOMM*, 2001, pp. 149–160.

[29] S. A. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," 2004.

[30] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in *Proc. ACM PODC*, 2002.

[31] B. Cohen, "Incentives build robustness in bittorrent," in *Proc. Wksp. on Economics of Peer-to-Peer Systems*, 2003.

[32] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice, "Dissecting bittorrent: Five months in a torrents lifetime," in *Proc. PAM 2004*, 2004, pp. 1–11.

[33] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *Proc. IPTPS'05*, 2005.

[34] Y. Cui, B. Li, and K. Nahrstedt, "ostream: asynchronous streaming multicast in application-layer overlay networks," *IEEE JSAC*, vol. 22, no. 1, pp. 91–106, 2004.

[35] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE JSAC*, vol. 20, no. 8, pp. 1456–1471, 2002.

[36] J. Keller and G. Simon, "Solipsis: A massively multi-participant virtual world," in *Proc. PDPTA 03*, 2003, pp. 262–268.

[37] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "Von: A scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.