FLoD: A Framework for Peer-to-Peer 3D Streaming

Shun-Yun Hu*, Ting-Hao Huang†, Shao-Chen Chang*, Wei-Lun Sung*, Jehn-Ruey Jiang * and Bing-Yu Chen‡
*Department of Computer Science and Information Engineering
National Central University, Taiwan, R.O.C.
†Department of Computer Science and Information Engineering
National Taiwan University, Taiwan, R.O.C.
‡Department of Information Management and Graduate Institute of Networking and Multimedia
National Taiwan University, Taiwan, R.O.C.

***Abstract*—Interactive 3D content on Internet has yet become popular due to its typically large volume and the limited network bandwidth. Progressive content transmission, or *3D streaming*, thus is necessary to enable real-time content interactions. However, the heavy data and processing requirements of 3D streaming challenge the scalability of client-server delivery methods. We propose the use of peer-to-peer (P2P) networks for 3D streaming, and argue that due to the non-linear access patterns of 3D content, *P2P 3D streaming* is a new class of applications apart from existing media streaming and requires new investigations.**

We also present *FLoD*, the first P2P 3D streaming framework that allows clients of 3D virtual globe or virtual environment (VE) applications to obtain relevant data from other clients while minimizing server resource usage. To demonstrate how FLoD applies to real-world scenarios, we build a prototype system that adapts JPEG 2000-based 3D mesh streaming for P2P delivery. Experiments show that server-side bandwidth usage can thus be reduced, while simulations indicate that P2P 3D streaming is fundamentally more scalable than client-server approaches.

I. Introduction

3D streaming refers to the continuous and real-time delivery of 3D content (e.g., meshes, textures, animations, etc.) over networks to allow user interactions without a prior download. Similar to audio or video *media streaming* [1], [2], 3D content needs to be fragmented into pieces at a server, before it can be transmitted, reconstructed, and displayed at the clients. However, unlike media streaming, because users accessing 3D content often have different visibility or interests, transmission sequence in 3D streaming thus varies from user to user and requires individualized visibility calculations [3].

Current 3D streaming schemes can be classified into four main types: object streaming, scene streaming, visualization streaming, and image-based streaming [4]. In this paper we look at *scene streaming*, which usually involves a collection of 3D objects placed arbitrarily in space that are streamed to clients according to user visibility or interests. The goal of scene streaming is to provide a *remote walkthrough* (i.e., navigation) or multi-user *virtual environment* (VE) experience [5], where users navigate a 3D scene and possibly communicate with one another in real-time (e.g., walkthrough of virtual museums). As many more objects may exist than what a user can see at a moment, scene streaming generally has two stages: *object determination* and *object transmission* [6]. 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 to send the object pieces [7]. Scene streaming also benefits from the reuse of cached content, so that objects need not be sent again if re-visited later [3].

In this paper, we try to answer the question: *how can 3D scene steaming be realized for millions of concurrent users within the same VE?* Existing 3D streaming schemes adopt the client-server architecture for content delivery. However, as 3D streaming is both data and processing-intensive, prohibitively vast amount of server-side bandwidth and CPU resources are required when serving a large audience. 3D applications with large data volume (e.g., today's popular *Massively Multiplayer Online Games*, or MMOGs [8]) thus currently require users to obtain the content through pre-installations via CDs or prior downloads. However, prior installations are undesirable and even unpractical for two likely future scenarios:

Larger and more dynamic content. Today's MMOGs have a few GBs of relatively static content (e.g., *World of Warcraft* is over 5 GB). However, as content becomes larger and more dynamic, streaming will save both the installation and update time. In fact, a prior installation is already unsuitable for the social MMOG *Second Life* [9], which depends on 3D streaming to deliver over 34 TB of user-created models, textures, and behavior scripts[1]. Also of note is that virtual globes such as *Google Earth* and *NASA World Wind* currently have terabytes of data (70 TB and 4.6 TB, respectively). Extensions into 3D may only be a matter of time, as shown by initiatives such as *X3D Earth*[2]. Pre-installation thus is unpractical given the size of the data and the userbase (i.e., 250 million+ *Google Earth* downloads). In realizing such *planet-scale virtual environments*[3], large-scale 3D streaming could be the basis for next-generation virtual globe applications.

Larger number of environments. As imagined by proponents of *Web 3D*, the future Internet could very well be three-dimensional where diverse environments exist to offer various socializing, shopping, and learning experiences. If *millions* of 3D sites were to exist, prior installations for each one of them would be frustratingly inconvenient and unpractical.

[1] http://www.informationweek.com/news/showArticle.jhtml?articleID=197800179
[2] http://www.web3d.org/x3d/workgroups/x3d-earth/
[3] https://www.technologyreview.com/Infotech/18911/

TABLE I
P2P STREAMING COMPARISONS

	live	on-demand	3D scene
starting position	same	arbitrary	arbitrary
access pattern	linear	linear	non-linear
trans. sequence	same	same (different starts)	unique
group switching	infrequent	infrequent	frequent

Scalable and efficient 3D streaming thus may be an important enabler for diverse forms of new 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 content due to overlapped visibility. Users thus might obtain relevant content from one another. Although P2P media streaming has seen significant progress in recent years, it is not directly applicable to 3D data due to the different access patterns. Consider that a user has left a certain navigation path through a VE, when other users also go through the same path with the same starting position and speed, the data stream would be the same for everyone (i.e., similar to *live media streaming* [2]). If other users join the path at different locations but proceed with the same speed, the data would resemble *on-demand media streaming* [1]. *3D scene streaming* occurs when other users proceed on different paths at different speeds, making the streaming sequences individually unique.

The main difference between 3D scene streaming and media streaming thus lies in the *content access pattern* due to user behaviors. Media streaming views content as being one-dimensional (i.e., time) and sequentially accessible, whereas 3D streaming views content as stored in a multi-dimensional space (i.e., the x and y axis, view orientation, etc.) and accessed according to user behaviors. The content access pattern thus is linear and more predictable for media streaming, yet non-linear and less predictable for 3D streaming [10]. This non-linear access pattern also makes dynamically forming and maintaining the proper *interest groups* that share data to enable P2P delivery much more challenging, as the switching between various groups occurs more frequently than in audio or video streaming (Table I). Novel understandings to the fundamental problems involved and the design of new streaming techniques thus are necessary for P2P 3D streaming.

This paper builds on our earlier work [4] to provide a conceptual model for 3D scene streaming, and presents the design and evaluation of *FLoD (Flowing Level-of-Details)*, the first P2P framework that supports 3D scene streaming for MMOG or virtual globe applications. By separating the graphics and the networking aspects of the problem, FLoD also allows both fields to tackle each aspect independently.

The rest of the paper is organized as follows: A model for P2P 3D scene streaming is first presented in Section II, followed by FLoD's design in Section III. To evaluate FLoD's applicability and scalability, Section VI and V describe a prototype system and related simulations. Section VI discusses related work, and conclusions are given in Section VII.

II. P2P-BASED 3D SCENE STREAMING

A. System Model and Assumptions

We consider a *remote walkthrough* [3], [11] 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, light maps, animations, etc., and the information on their placements is stored within a *scene description*. Each user navigates the scene through a client program and may update his or her current position and view orientation via movement commands (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 or interact with all of them at once. Each user's visibility and interaction thus is limited to a circular *area of interest* (AOI) [5] centered at the user's current location. For simplicity, we assume that all objects are static in both their positions and content. In this basic model, we also do not consider the display of other users' 3D representations (i.e., each user sees only 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*. The specific fragmentation is application-specific, 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 to allow navigation. *Progressive meshes* [7] and techniques such as *geometry image* [12] may be used for mesh fragmentation, while progressive encodings of GIF, JPEG, or PNG, may be used for texture fragmentation [13]. Note that different fragmentation methods impose different types of dependency requirements among the pieces when reconstructing the objects.

All content is initially stored at a server, and clients obtain it through streaming 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.

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* [3] or *visual perception* [11]. However, as visual quality can be subjective, 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 100% indicates the best visual quality, as the rendered image would be the same as if all content is 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*, the time to download the complete data of an object. Note that the two are similar to *latency time* and *response time* in [11]. Base latency indicates the delay for a user to see a basic view of an object, while completion latency indicates the delay of being

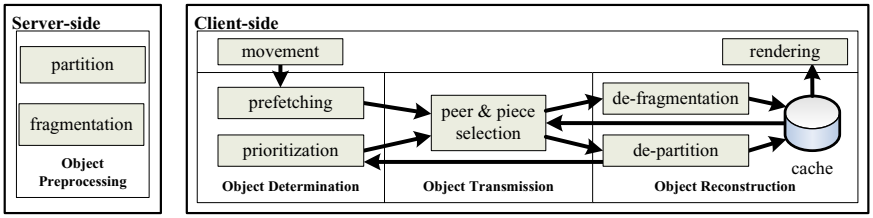


Fig. 1. A conceptual model for P2P-based 3D scene streaming.

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 content is 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 three new issues to address:

Distributed visibility determination: Preferably, visibility should be determined without the server's involvement or any global knowledge of the scene. However, as only the server initially has complete knowledge on object placements (i.e., the *scene descriptions*), we need to partition and distribute scene descriptions to clients so that visibility can be determined in a distributed manner efficiently.

Dynamic group management: The key that the server's load may be reduced is to let clients form data exchange groups based on common interests in the 3D data. This involves the efficient discovery and maintenance of such interest groups among clients. Note that as users are moving constantly, the grouping is much more dynamic than that in media streaming.

Peer and piece selection: Once a client has known a group of peers where common interests exist, to optimize the visual (or streaming) quality for a given bandwidth budget, the client 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. Interestingly, as 3D streaming is view-dependent [14] and that some data pieces may be applied in arbitrary order during object reconstructions, 3D streaming requires only a *roughly sequential* transfer order as opposed to the *strictly sequential* transmissions in video or audio streaming, as long as certain piece dependencies are satisfied.

D. Conceptual Model

Given the above requirements and challenges, we summarize the main tasks for P2P 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 be 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 examples of fragmentation techniques.

Prefetching: The task of predicting data usage ahead of time and generating object or scene requests so that latency due to transmissions is masked from users. Predications of user movements or behaviors are often employed for this task [11].

Prioritization: The task of performing visibility determination to generate the ordering for a client to obtain object pieces in a scene. The goal is to produce the best streaming quality with considerations to factors such as object distance, line-of-sight [3], [11], or the requesting client's bandwidth [15].

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 efficiently fulfill requests from prefetching and prioritization.

Fig. 1 organizes the above tasks into a conceptual model for P2P-based 3D scene streaming. For an interactive 3D application, obtaining *movement* updates from the user and performing *rendering* are the only steps when content is locally available. *Object preprocessing, determination, transmission*, and *reconstruction* are the additional stages in 3D streaming. For client-server-based 3D streaming, only *fragmentation*, *prefetching*, and *prioritization* need to be considered. *Partition* of the scene and the *selection* of peers and pieces are new issues introduced in P2P-based 3D streaming. A summary comparison between the two approaches is shown in Table II.

TABLE II
CLIENT-SERVER AND P2P COMPARISONS.

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

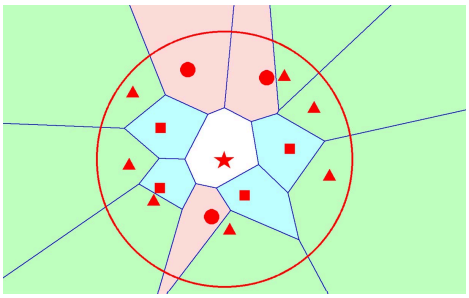


Fig. 2. Neighbors in VON: *boundary*(triangle) *enclosing*(square) both(circle)

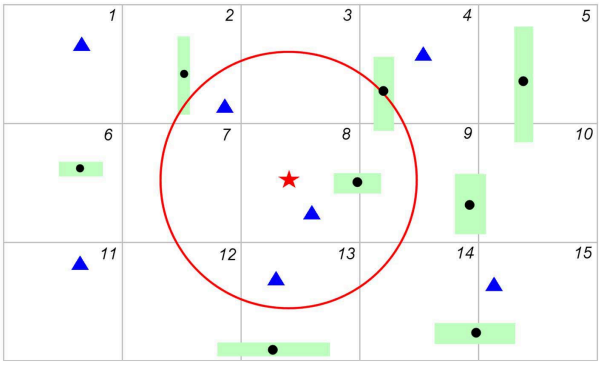


Fig. 3. Schematic of a virtual environment divided into *cells*.

III. Design of FLoD

A. Overview

FLoD's main design rationale is that as users in large-scale VEs tend to see each other or crowd at certain *hotspots* [8], a node might have overlapped visibility with its *AOI neighbors* (i.e., other nodes whose positions fall within the node's AOI). It is thus likely that the neighbors already possess relevant 3D content. By requesting data from the neighbors first, the server can be relieved from serving the same data repetitively. Note that neighbors here are based on proximity on the virtual map, not the physical network. The discovery of AOI neighbors is in fact the discovery of the proper interest groups for distributed content sharing, and must be done efficiently (i.e., the challenge of dynamic group management). Fortunately, recent research on *P2P virtual environment* (P2P-VE) overlays [8] allows information on AOI neighbors such as IDs, virtual coordinates, and IP addresses be learned given a position and AOI-radius, without relying on a server. As a node moves around, it can constantly notify the overlay of its position and get refreshed information on AOI neighbors.

Our choice for the P2P overlay is *Voronoi-based Overlay Network* (VON) (Fig. 2), as it has demonstrated scalability, consistency, and reliability [8]. VON requires each node to connect directly with its AOI neighbors and organize them into a *Voronoi diagram*. By identifying the *boundary neighbors* (i.e., nodes whose Voronoi regions overlap with the AOI boundary), a node may learn of new nodes from its boundary neighbors as it moves around. To further constrain client-side bandwidth usage, a node may also shrink its AOI if a certain *connection limit* is exceeded [8]. Note that other P2P-VE overlays can also be used [8], as long as correct and timely information on AOI neighbors are provided. One benefit of VON is that when no AOI neighbors are present, connections with a few *enclosing neighbors* are still kept [8], such that data requests to peers are still possible.

To efficiently distribute scene descriptions to clients (i.e., the challenge of distributed visibility determination), we partition the VE into fixed-size square *cells* (similar to *Cyberwalk* [11]), 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. Determining the visible objects to retrieve can thus be done in a fully distributed manner, as each node is able to locally determine the cells covered by its AOI (see Fig. 3, where the big circle is the AOI of the star node, and triangles are other user nodes. Various shapes are the 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). When entering a new area, a client first prepares a *scene request list* to obtain scene descriptions from its AOI neighbors or the server. Once scene descriptions are obtained, the client then judges which objects are in view and produces a *piece request list* to request for visible objects. Piece dependency, if any, is also specified in the piece request list to ensure that data retrieval adheres to the correct piece ordering for object reconstructions. Views are rendered progressively as data pieces arrive from either the peers or the server (which acts as the final data source if peers cannot fulfill the requests). This iterative process of requesting scene descriptions and object pieces is repeated continuously as a client moves in the VE.

To accommodate evolving policies and techniques (i.e., the challenge of peer and piece selection), FLoD separates 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 is not yet considered in this work, but is included for the sake of completeness. The *application* sits on top of FLoD and performs the usual tasks of taking user *movement* commands and performing *rendering*.

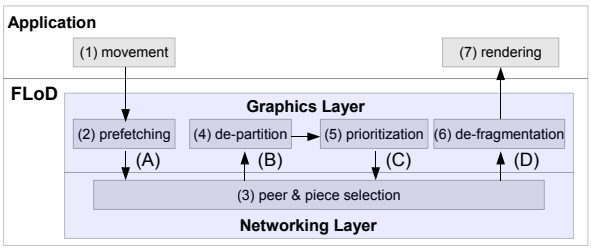


Fig. 4. FLoD's client-side task flow and layers. Data flows: (A) scene request list (B) scene descriptions (C) piece request list (D) data pieces. The numbers are task labels in FLoD's *Procedures*.

B. Procedures

We now describe FLoD's main procedures in more details. The numbers after the procedure names indicate the tasks (as shown in Fig. 4) covered by the respective procedure:

Login: The joining node enters the VE system by specifying a join location and AOI-radius to the P2P-VE *overlay*, which returns an initial list of AOI neighbors. The VE'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 *scene request list* 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 *piece request list*, 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 the specified location, orientation, and scale of each object in the *scene descriptions*.

Request for Data (3): If the local cache does not have 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 peers fail to respond.

Move (1): A node moves by sending a user-generated position update to the *overlay*, which forwards the update to other AOI neighbors. Any new neighbors discovered via the overlay will become part of the *data source nodes*. If the node enters certain new cells whose *scene descriptions* are unknown, *Obtain Scene Descriptions* is invoked.

Logout: A node simply disconnects from all neighbors when leaving the system. As the system is distributed, failure or departure of any single user node will not affect the system's operation. Other nodes will learn about the departure from the *overlay* via an updated neighbor list.

C. Policies

The above procedures describe the general steps when browsing a 3D scene. However, specific policies are still needed for various streaming tasks, which are discussed below:

Content Discovery: Before each peer can request data, they must first know which neighbors possess the desired content. Some methods include: 1) request data from neighbors simultaneously; 2) request data from neighbors sequentially; 3) query the neighbors first, and send requests later. The first option has the least latency, but is vulnerable to multiple responses. The second option uses the least bandwidth, but incurs more delays due to multiple attempts. We therefore query the neighbors first, and only request the data from the neighbors that respond positively.

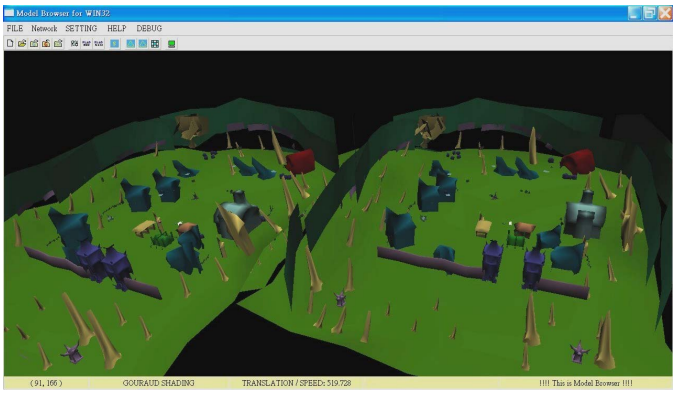


Fig. 5. Screenshot of the 3D streaming client.

Peer Selection: Once a peer knows which other neighbors possess a certain data piece, it could send a request to a chosen peer. The choice of peer can either be random or based on certain criteria (e.g., remote peer's bandwidth capacity or proximity). Our current choice is to pick neighbors randomly.

Piece Selection: For obtaining pieces, the request order need not be strictly sequential as the piece dependencies for 3D objects may follow that of a tree (e.g., geometry image [12]) or a forest (e.g., progressive meshes [7]). For simplicity, we now adopt a sequential request policy for data pieces. Note that even though individual objects are retrieved linearly, as the set of requested objects changes constantly, the overall content access pattern is still non-linear.

Server Request Condition: One key question for P2P delivery is *under what conditions do clients request data from the server?* The answer impacts both the number of requests to a server, and the responsiveness for clients to obtain data. A basic approach is to ask the server whenever other clients cannot respond to requests. However, this could make the server vulnerable to requests, especially when clients are joining concurrently. We thus allow a client to request from the server only if it becomes the nearest node to an object.

Caching: We choose a cache size that will store roughly three times the expected amount of data within an AOI. The farthest objects from the user's position are replaced first when the cache is used up.

IV. Prototype Implementation

To demonstrate how FLoD applies in actual scenarios, we implement a prototype based on geometry image streaming [12], which converts an arbitrary 3D mesh model into a 2D image, so that progressive mesh streaming is achieved by sending the 2D image. Fig. 5 shows a screenshot of our 3D streaming client, where two scenes are shown. In this section, we describe the partition, fragmentation, prioritization, selection, and object reconstruction methods used by our prototype. Prefetching as noted earlier is not yet considered.

Partition: As publicly available large scenes are hard to find, we use a small virtual village (a 3D Studio Max scene file) from an actual game demo as the basic cell and convert it into a X3D file. 3D objects are extracted from the X3D file and

TABLE III
PROTOTYPE LAN EXPERIMENT STATISTICS

	total time (ms)	send size (b)	recv size (b)	base latency (ms)	SRR	avg AOI neighbor
min.	10,188	6,376	397,074	9	0.000	0.583
max.	63,053	5,792,518	8,232,699	1,250	0.975	5.000
avg.	36,064	1,006,912	1,349,224	502	0.366	3.192
std. dev.	17,569	1,126,383	1,145,514	395	0.269	1.073

converted to geometry images [12], so that the X3D file serves solely as the scene description. For the test scene, we duplicate the village 100 times by translating and tilting randomly positioned objects to fit on a larger plane, until the total data volume approximates that of a real game scene. The original data for a cell is 514 KB (with a scene description and models), and the final VE is 50.5 MB. As MMOG developers often build scenes in cell-based units [9], our approach resembles at least in part with how MMOGs or VEs are constructed today.

Fragmentation: As the scene descriptions store only the IDs and bounding boxes of each objects, we use JPEG 2000 to store the geometry images of the actual models as it is a public standard for image-based streaming. To facilitate progressive transmissions, the images are fragmented into pieces with JPIP [12] in the following way: 1) Resolution: as rendering distant objects with many polygons is unnecessary, we divide each image into several *resolution levels*, where only the appropriate levels are used to render a scene. Each level, except the highest one, represents a simplified version of the original 3D model. 2) View-dependency: when a user looks at the front of an object, transmission of the rear-side data can be postponed. Each resolution level thus also divides into *blocks* that correspond to different surface patches on the reconstructed model. In our prototype, each image is divided into 5 resolution levels. Level 0 corresponds to the base piece and all other blocks correspond to the refinement pieces.

Prioritization: As mentioned in Section III-B, after joining the P2P network, each node first requests the scene descriptions to perform distributed visibility determination. After scene descriptions are obtained, prioritization then generates a *piece request list* for locally unavailable data. The intention is to retrieve each object in its *optimal resolution* in respect to the user's perception requirements. Here we adopt the concept of *visual importance* in *Cyberwalk* [11] for the optimal resolution, where higher visual importance is given to objects nearer to the viewer or closer to the center of the field of view. However, some differences exist between our method and *Cyberwalk*: 1) Since each client has the scene description, visual importance is calculated by each client instead of the server, reducing the server's load. 2) The number of data requests for a particular object is based on both bandwidth utilization estimates and its bandwidth quota (based on the object's visual importance), so that more important objects can utilize a larger share of the available bandwidth.

Selection: Once the network layer receives the piece request list, peer and piece selections proceed to fulfill the requests. We rely on asking the AOI neighbors to provide the relevant 3D content. For each piece on the request list, a query is first sent to all AOI neighbors to check for data availability. Actual requests for the data pieces are sent to a randomly chosen neighbor that responds positively. For a given neighbor, at most five requests can exist at a time. New requests are sent only after previous ones have finished, so that neighbors with higher transfer rates can service more requests. If none of the neighbors has a desired piece, the requester will keep querying until the server request condition (see Section III-C) is met.

Object Reconstruction: To reconstruct a model for rendering, we expand the JPEG 2000 image from cache to a certain resolution level according to the object's visual importance. When a user node moves close to the current cell's boundary, the node will request new scene descriptions in the nearby cells, repeating both prioritization and selection.

V. EVALUATION

We perform two main types of evaluation for FLoD: the first is an actual session of running the prototype on a LAN with multiple users. Such experiment demonstrates the feasibility of using P2P streaming to save server resources. However, to see how FLoD works on a larger scale requires simulations. The main purpose of the simulation is to compare the *scalability* and *streaming quality* between a P2P and a client-server approach to 3D streaming. In this section, we present our performance metrics, LAN experiment, followed by the simulation setup and simulation results regarding the scalability, streaming quality, and limitations of FLoD.

A. Performance Metrics

We adopt the following metrics in evaluating FLoD:

Bandwidth usage: A 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 content. We can 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 can start meaningful navigation when entering a new scene.

Cache utilization: To see how cache content may be reused to serve requests from other peers, we record the distribution of the number of reuses for each data piece. This shows how node density or movement models may affect cache use.

B. LAN Experiment

Our experiment with the prototype involves setting up a server that loads the initial scene data onto memory, and responds to client requests as needed. To conduct the experiment, ten computers on a 100 Mbps LAN are setup to act as clients. During a 40-minute session, users login to the system and navigate around continuously to explore the scenes. Statistics on 48 sessions regarding the clients' performances are collected and shown in Table III.

As can be seen from the data, each client stays within the scene for roughly 36 seconds per session, and has three known neighbors on average. As clients could request data from their neighbors, the average *server request ratio* (SRR) (i.e., the ratio of the received data that comes from the server) is about 36.6%. The average upload and download transmissions of clients are roughly the same, indicating that most clients do well to serve other peers of their data needs.

C. Simulation Setup

A custom discrete-time simulator is used for our simulations, which proceed in *time-steps* of 100ms each. Up to n nodes are put inside the simulator to process and exchange messages with other nodes at each step, under the following bandwidth limits: 1 Mbps download and 256 Kbps upload for typical broadband clients and a 10 Mbps symmetric connection for the server. Constant latency is also assumed between all nodes, where each message sent can be received in the next step, unless the transmission time is prolonged due to bandwidth limits. The simulator runs on top of VAST, an implementation of the P2P-VE overlay VON (http://vast.sourceforge.net/). To constrain bandwidth used by the overlay, a connection limit is also set for each node so to restrict the number of connected AOI neighbors [8].

To set up a VE, a number of objects are randomly placed on a 2D map that is partitioned into square cells. For simplicity, we assume that each object has only one set of data pieces. Based on the size of the data used in our prototype, each object is set to 15 KB, where the base piece is 3 KB and 10 refinement pieces are 1.2 KB each. The scene descriptions are around 300 to 500 bytes each. To run the simulation, a number of nodes are put randomly within the VE, and stay at their joining locations until the the system's average fill ratio exceeds 99%. This gives each node an initial set of data to share. The nodes then move with constant speeds, and request scene descriptions or data pieces as needed. We adopt both a random way-points [11] and a clustering movement model to see how a uniform and a clustered distribution of nodes would affect FLoD's performance. Clustering movement is done by randomly placing 1.5 * $ln(n)$ hotspots (n is node size, so 100 nodes has 6 hotspots) where nodes would move towards the nearest hotspot with high probability. All simulations proceed for 3000 steps, which is equivalent to 300 seconds assuming 100ms per step. As we are interested in the system's stable state behavior, the rest of the discussions will be based on statistics collected during each simulation's last 2000 steps. Specific simulation parameters are shown in Table IV.

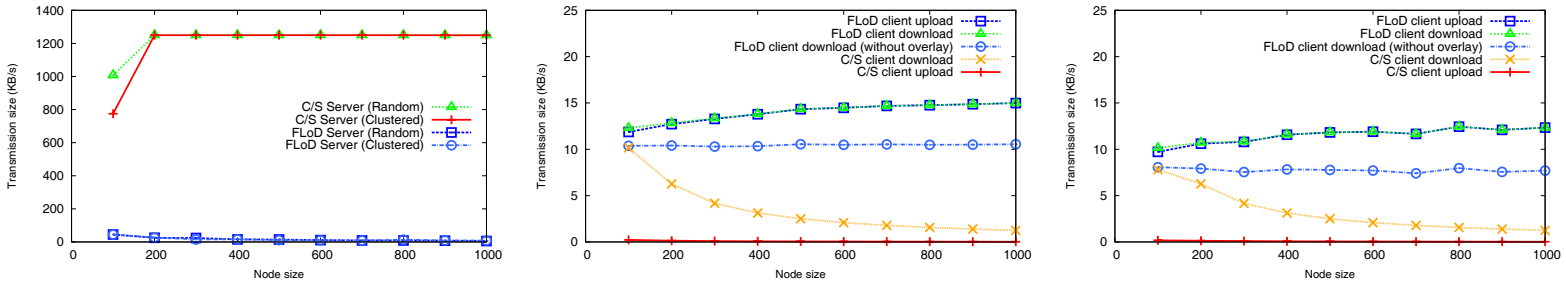
TABLE IV
SIMULATION PARAMETERS

World dimension (units)	1000x1000
Cell size (units)	100x100
AOI-radius (units)	75
Overlay connection limit	14
Time-steps	3000
Number of nodes	100 - 1000 (in 100 increment)
Number of objects	500
Node speed (units / step)	1
Client cache size (KB)	400

D. Simulation Results

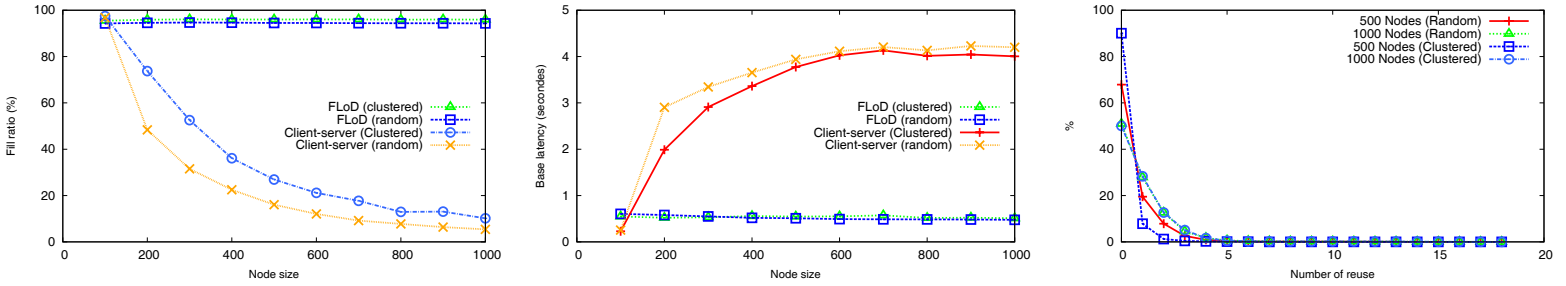
Scalability: The basic requirement for scalable systems is that resource usages should be bounded at all relevant system nodes. In the context of a streaming system, it means that both the server's and clients' bandwidth usage should be bounded by some limits. Fig. 6(a) shows the upload bandwidth for both a C/S server and a FLoD server under both movement models. As the bandwidth limit is 10 Mbps for the server, the C/S server's bandwidth exhausts at 1250 KB/s (i.e., 10 Mbps) when serving over 200 nodes. On the other hand, a FLoD server's upload stays relatively constant below 50 KB/s (i.e., 400 Kbps). The reduction in server-side bandwidth is explained in Fig. 6(b) and Fig. 6(c), which show the convergence of the upload and download bandwidth of FLoD clients, indicating that as the system scales (i.e., the number of AOI neighbors increases), FLoD clients can become self-sufficient in mutually serving data. On the other hand, C/S clients are rationed less and less of the server's bandwidth, and their download sizes continuously decrease. Note that both random and clustering movement models produce similar overall patterns, with the difference that the clustering movement model utilizes less client bandwidth (as nodes tend to stay near *hotspot* locations, the demand for new content thus lessens), and produces a more fluctuating usage pattern. The results also show that P2P 3D streaming is feasible under today's broadband environment given our assumed content size and user behavior.

Although FLoD significantly reduces and bounds server-side bandwidth usage, client-side bandwidth is still consumed logarithmically as the number of users increases, which indicates that the system still has a scalability limit. However, additional analysis reveals that the increase is mostly due to the P2P-VE overlay, whose bandwidth usage grows logarithmically with node density given a connection limit [8]. Bandwidth used by FLoD in fact remains relatively constant, which should be expected given our uniform object distribution and constant node speed (i.e., the new content required by each node per-second remains constant on average). As the bandwidth growth depends on user density, it means that if it can be controlled, the *total* number of supportable users can grow scalably. Note that 1000 nodes in a 1000x1000 area is already a high user-density scenario. This shows that P2P-based 3D streaming is fundamentally more scalable than client-server approaches by preventing both the server and clients to become resource bottlenecks.



(a) server upload (random/cluster movement) (b) client upload/download (random movement) (c) client upload/download (cluster movement)

Fig. 6. Bandwidth usage comparisons (average transmission size per node per second).



(a) fill ratio (b) base latency (c) cache utilization distribution

Fig. 7. Streaming quality comparisons and cache utilization.

Streaming Quality: We use *fill ratio* and *base latency* to measure the streaming quality of the system. Fig. 7(a) shows the fill ratio for both C/S and FLoD clients. After an initially high fill ratio (while the server still has bandwidth), the ratio of C/S clients drops at 200 nodes (i.e., 73.72% for clustering, and 48.36% for random movement), and then declines continuously. On the other hand, a FLoD client's fill ratio is relatively stable regardless of node size (i.e., over 94% for both movements). The degrade in the service quality for C/S clients is also seen from Fig. 7(b), where the base latency is initially lower than FLoD clients at 100 nodes, but increases significantly afterwards. On the other hand, FLoD clients' base latency is relatively stable at below 600ms. We also look at how well the cache at each client is utilized to support other peers. Fig. 7(c) shows that cache is better utilized for denser nodes (at 50% utilization) and for random movements.

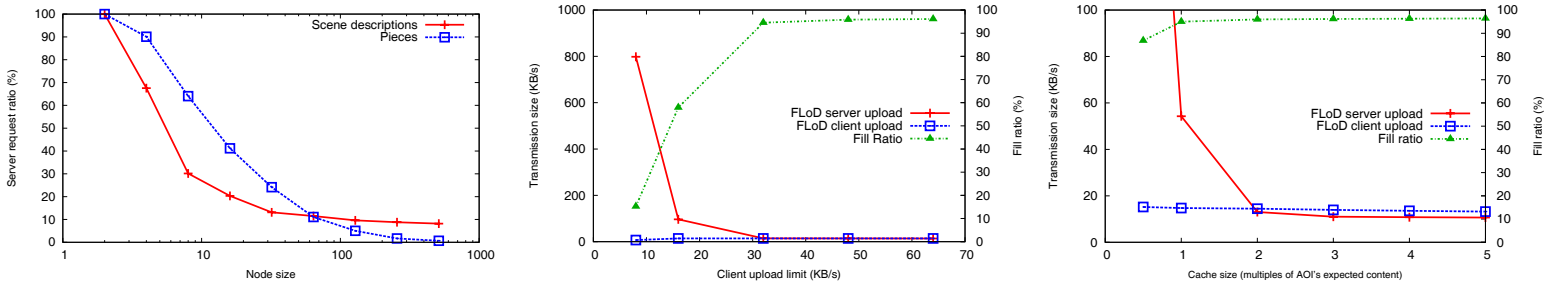
Limitations: FLoD assumes that clients can obtain data from neighbors with shared visibility, an interesting question thus is *what happens if AOI neighbors do not exist?* To answer, we perform another set of simulations with random movement to observe how the server request ratio might change as node size varies from 2 to 512, while fixing all other parameters. Fig. 8(a) shows how server request ratio may decrease as node density (and hence the number of AOI neighbors) increases. When few AOI neighbors exist, most requests will go to the server and be fulfilled in client-server fashions. This shows that FLoD needs sufficient AOI neighbors to function properly.

Even when AOI neighbors exist, *what happens if the amount of data required exceeds what the peers can provide?* In another experiment with 500 nodes and random movement, we downgrade clients' upload from 64 KB/s to 48, 32, 16 and 8 KB/s (i.e., 512, 384, 256, 128, 64 Kbps, respectively). Note that by lowering the upload capacity, we are looking at the effect of *data density* on FLoD. Fig. 8(b) shows that the fill ratio remains high till 32 KB/s (i.e., 256 Kbps), but decreases as the client upload gets smaller. This indicates that the peers' uploads must exceed the amount of data needed by peers, otherwise either the server would receive excessive requests, or the service quality for peers would degrade.

In another set of 500-node / random movement simulations, we see how cache size affects the streaming quality and bandwidth usage by varying cache sizes between 0.5 to 5 times of the expected content in an AOI (estimated to be 132 KB on average). Fig. 8(c) reveals that the fill ratio degrades and the server's bandwidth usage surges if the cache is less than one AOI's content. The cache is adequate if it is twice the AOI content, but fill ratio and bandwidth usage improve only little beyond that, indicating that excessive cache is not necessary.

VI. Related Work

Schmalstieg and Gervautz [16] first introduce scene streaming where a server determines and transmits visible objects at different *level-of-details* (LODs) to clients. Subsequent work replaces discrete LODs with continuous (*smooth*) LODs [17]. Teler and Lischinski used pre-rendered image-based *impostors* as the lowest LOD to allow faster initial visualizations [3]. *Cyberwalk* [11] adopts progressive meshes to avoid the data redundancy from multiple LODs, and focuses on caching and prefetching to enhance visual perceptions. Deb and Narayanan



(a) effect of node density (b) effect of upload capacity (c) effect of cache size

Fig. 8. Limitations of FLoD.

propose a *geometry streaming system* that maintains interactive frame-rate by adaptive data selection [15]. Social MMOGs such as *ActiveWorlds*, *There.com*, and *Second Life* [9] utilize scene streaming to support dynamic content, but little is known on their mechanisms. Our work complements the above work with distributed deliveries. Mayer-Patel and Gotz present the concept of *non-linear media streaming* [10] where interactive content (e.g., images for a virtual museum) is divided and sent through some multicast channels subscribed by clients. The system scales by using application-layer multicast. However, ensuring proper content partition and bounded latency (important for interactive applications) are non-trivial. FLoD considers actual 3D content, delivers only the content that users need, and performs content discovery with bounded delay. *Level-of-detail description tree* [18] is introduced recently to determine visibility for hierarchically-organized urban scenes. A few peer selection strategies based on proximity or estimated content on peers are also evaluated. However, its performance as user size scales has not yet been reported.

VII. Conclusion

We have formulated a conceptual model for P2P-based 3D scene streaming by identifying its main tasks and presented the first related framework where a P2P-VE overlay is used to discover neighbors for content exchange. We show FLoD's feasibility with a prototype, and how bandwidth usage is bounded to achieve scalability. An open source implementation of FLoD is available at: http://ascend.sourceforge.net/.

A number of directions exist for future work, for example, the current design requires sufficient AOI neighbors, yet nodes beyond AOI may also possess relevant content that can be considered as sources. We assume linear piece dependency, yet non-linear dependency may provide better download parallelism. We also have not investigated prefetching in depth, but it is essential for any streaming scheme to be effective.

Real-time 3D content has 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 a topic of interest to both graphics and networking professionals. By identifying the basic issues, we hope to generate interests in this promising direction for more accessible 3D content.

Acknowledgments

This work was supported by NSC, Taiwan, R.O.C. under 95-2221-E-008-048-MY3 and 95-2221-E-002-273-MY2. We thank Prof. Shing-Tsaan Huang for his supports, Nein-Hsien Lin for the 3D client, Guan-Yu Huang for the prototype server, and Actainment Co. (http://www.actainment.com/) for the game scene. We are also grateful of the facilities by LSCP, Academia Sinica, the National Center for High-performance Computing, and comments by the anonymous reviewers.

References

[1] 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.

[2] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in *Proc. INFOCOM*, 2007, pp. 1415–1423.

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

[4] S.-Y. Hu, "A case for 3d streaming on peer-to-peer networks," in *Proc. Web3D*, 2006, pp. 57–63.

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

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

[7] H. Hoppe, "Progressive meshes," in *Proc. SIGGRAPH*, 1996.

[8] 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.

[9] P. Rosedale and C. Ondrejka, "Enabling player-created online worlds with grid computing and streaming," Gamasutra Resource Guide, 2003.

[10] K. Mayer-Patel and D. Gotz, "Adaptive streaming for nonlinear media," *IEEE Multimedia*, vol. 14, no. 3, pp. 68–83, 2007.

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

[12] N.-S. Lin, T.-H. Huang, and B.-Y. Chen, "3d model streaming based on jpeg 2000," *IEEE TCE*, vol. 53, no. 1, 2007.

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

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

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

[16] D. Schmalstieg and M. Gervautz, "Demand-driven geometry transmission for distributed virtual environments," *CGF (EG 1996)*, vol. 15, no. 3, pp. 421–433, 1996.

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

[18] J. Royan, P. Gioia, R. Cavagna, and C. Bouville, "Network-based visualization of 3d landscapes and city models," *IEEE CG&A*, vol. 27, no. 6, pp. 70–79, 2007.