Streaming and Synchronization of Multi-User Worlds Through HTTP/1.1

Jean-Eudes Marvie* Pascal Gautron† Pascal Lecocq ‡ Olivier Mocquard § François Gérard ¶
Technicolor Research & Innovation

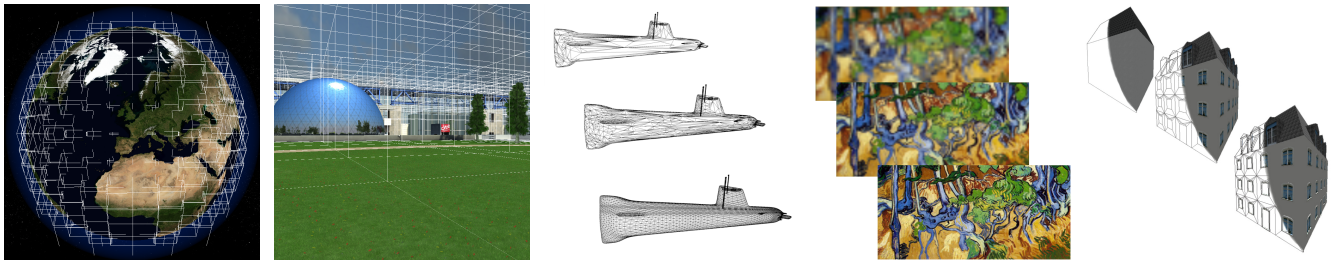

Figure 1: *Our approach for adaptive streaming at multiple levels: planet-scale structuration, visibility-based occlusion culling, multi-resolution meshes and textures, and adaptive reconstruction of procedural models.*

1 Introduction

As the trend of online multi-user worlds gets more and more momentum, such worlds usually require heavy infrastructures both in terms of hardware and software: servers often manage the entire world simulation, and hence limit the number of simultaneous connections. The data exchanges are performed using proprietary protocols, requiring specific server applications and the use of dedicated ports which leads to potentially complex proxy issues for connection. Also, online virtual worlds usually target specific platforms (e.g. PC for Second Life, or game stations for Playstation Home), and even reduce the use of the world to a subset of available platforms due to bandwidth or hardware requirements.

We propose an approach for adaptive streaming of online multi-user virtual worlds, using generic transfer protocols and a unified representation of the worlds usable on an arbitrary wide range of platforms. HTTP/1.1 natively supports two crucial features for data streaming: persistent connections and chunked access to files. We leverage those capabilities to avoid the need for specific server software, communication ports and transfer protocols. Also, a side effect of the use of HTTP/1.1 is an enhancement of the usability: most current proxies support this protocol, hence sparing the user a specific tuning of proxies and firewalls.

Based on those simple capabilities we propose unified representations of arbitrarily large virtual worlds based on X3D, as well as methods for adaptive streaming and interaction regarless of the network bandwidth or capabilities of the client hardware.

2 Overview

Our main target is a client-independent representation of the virtual worlds, client-adaptive data streaming and decentralized multi-user interaction. To this end, we first address the problem of the efficient representation of planet-sized virtual worlds, bearing in mind the constraints of HTTP/1.1 [Fielding et al. 1999]. As shown in Figure 1 we first use a global structuration for the largest features of the world such as the planetary terrains. At a smaller scale, efficient occlusion culling is performed using cell-based space partitioning. Visible objects are then represented as multi-resolution meshes and textures. Also, further optimizations are obtained using procedural modeling of specific object types such as buildings or plants.

Our approach is based on a click-and-play approach: we focus on direct data streaming to avoid the need for prefetching a large amount of data before entering the virtual world. To ensure sufficient reactivity on data fetches, the servers can be preferably located within the physical neightborhood of the users.

Once the virtual environment is represented, we consider the immersion and interaction of multiple users. Unlike most existing approaches, we relieve the server from the synchronization and simulation tasks. To this end, we designed a novel X3D node for a decentralized peer-to-peer synchronization, potentially dependent on the geographical proximity of the users *in the virtual world*: regarless of their physical location in the real world, each user must be synchronized with the users within a close neighborhood in the virtual world, while potentially neglecting synchronization with more distant users for performance (Figure 2).

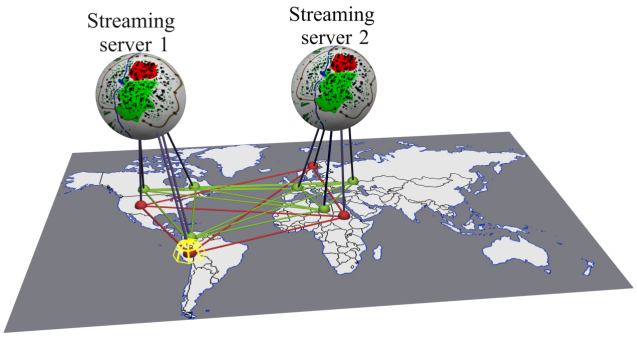

Figure 2: *Overview of our synchronization strategy: the data representing the virtual world are streamed by geographically close servers, while users in the same neighborhood (here, red and green zones) of the virtual world are connected on a peer-to-peer basis. One or more users can also take the role of yellow pages server to allow newcomers to connect the virtual world. All the communications and transfers are performed through HTTP/1.1.*

*e-mail:jean-eudes.marvie@technicolor.com
†e-mail: pascal.gautron@technicolor.com
‡e-mail: pascal.lecocq@technicolor.com
§e-mail: olivier.mocquard@technicolor.com
¶e-mail:francois.gerard@technicolor.com

3 World delivery

The usual delivery scheme for online virtual worlds starts with a massive download of resources files, including most of the 3D models, as well as texture maps. Before entering the virtual world, the user must not only wait for the necessary download to be performed, but also have a sufficient amount of available storage. While current desktop computers usually contain very large hard disks, this is not the case for smaller clients such as handheld devices. Also, this pre-downloaded resource package must contain the resources representing the entire virtual world. In the context of high quality, planet-sized worlds, the size of the package becomes prohibitive even using broadband connections and high-end computers. This delivery scheme is typically used in World of Warcraft[1].

If the virtual world targets several platforms (e.g. PCs, tablets, mobile phones, etc.), those problems are partially overcome by providing resource packages adapted to each platform. For example, a package targeting low-end handheld devices would feature degraded textures and models, while high-end gaming platforms would benefit from a larger, more detailed dataset. However, this approach requires replicating resource packages for each target platform, potentially leading to massive storage waste on server side. Furthermore, the dataset is not dependent on the current available bandwidth: for example, let us consider a high-end laptop computer connected using a 3G key. The user needs to either take a very long time to download a resource package adapted to his hardware, or access the virtual world more rapidly by choosing a degraded representation of the world. In this latter case, the user must re-download a larger resource package when eventually getting access to a higher bandwidth such as a WiFi connection.

The issue of resource availability can be addressed in several ways. The approach used in Second Life[2] is designed for a world subdivided into independent islands. Before entering an island the client downloads a minimal dataset corresponding to the objects visible from the entry point in the island. Further data are downloaded afterwards during navigation regardless of visibility or navigation. The simulation of the interaction between users is performed on server-side, hence limiting the number of simultaneous clients on a given island.

In [Cavagna et al. 2006; Royan et al. 2007] the virtual world is uncentralized, i.e. entirely distributed among the participants and available using a peer-to-peer approach. The lack of centralized server allow the users to create persistent virtual worlds without the need of dedicated infrastructure. However, problems may arise due to the high latency of P2P approaches. Furthermore, the upload link on client-side tends to be small (e.g. 1Mbps) on consumer connections.

Our approach combines P2P and server-based storage: the resources describing the world are stored on the server, while the interaction and synchronization are performed on client-side using a P2P approach. Instead of downloading resources beforehand, we use adaptive streaming to provide the user with a click-and-play experience at all levels: a global structuration of the world, in the spirit of X3D Earth, provides a progressive loading scheme for parts of the virtual world. At a finer level, we propose using an accurate management of visibility using cell-based Potentially Visible Sets. Finally, based on available bandwidth, memory and visual importance of objects, we leverage continuous levels of details for 3D meshes as well as multi-resolution texture maps and adaptive procedural reconstruction for a complete adaptive delivery pipeline for virtual worlds (Figure 3).

3.1 Global structuration

The virtual world is structured differently depending on the current distance between the ground and the user. As shown in Figure 4 the main elements of the ground infrastructure, which are visible from large distances, are represented using discrete levels of detail stored in separate files. This LOD management builds upon the Geospatial component of the X3D specification, and is detailed in Section 3.1.1. When the viewpoint gets closer to the terrain, another structuration based on visibility accounts for highly occluded environments such as cities (Section 3.1.2). The terrain itself is represented using the progressive techniques described in Section 3.2.

3.1.1 X3D Earth

The global structuration for streaming purposes could be performed using `Inline` nodes dynamically combined using scripting. To improve both design workflow and computational efficiency, as well as numerical precision, this structuration can be more easily performed using the Geospatial component of the X3D specification, also known as X3D Earth. This component targets the representation of planet-scale GIS data as well as navigation, user tracking and rendering. The `GeoLOD` node provides a quadtree-based representation of a terrain using several levels of detail. Each node of the tree stores the URL of the file containing its corresponding level of detail and the URLs of files describing finer levels of the tree. The files are loaded on-demand using HTTP file reads depending on the distance between the nodes and the current viewpoint. The structure also discards distant and invisible nodes, leading to a global stream consumption over the planetary surface.

If we consider a full-featured virtual world seen from the ground level, the approach introduced in X3D Earth would trigger file loads for every node in the neighborhood of the user, regardless of the actual visibility of the nodes. If the models comprised within those nodes are highly detailed, the navigation would be frequently stopped or incomplete while loading the fully detailed description of elements of the world even though they are currently invisible. Besides the loss of interactivity or visual quality, such detailed models may also not fit in the memory of the client.

Based on those observations, we use a finer structuration level based on visibility relationship between smaller elements of the virtual world so that node fetches are only issued advisedly.

3.1.2 Visibility structuration

When the viewpoint is located near the ground level, we switch the world representation to optimize the data streaming and rendering performance using visibility information. To this end, we subdivide the scene into a grid of *cells* as shown in the bottom level of Figure 4. In addition of the list of objects contained in the cells, each cell also carries information describing the set of neighboring elements which are potentially visible. More precisely, each cell stores its set of potentially visible cells, a set of potentially visible objects and the set of objects actually contained within the cell. The cells are represented as a separate files, and can be fetched using regular HTTP file requests.

This space partitioning can be easily performed using X3D extensions such as [Marvie and Bouatouch 2004]. This extension has the advantage of minimizing the number of file downloads required to render the world from a given region in space.

[1] http://battle.net/wow

[2] http://secondlife.com/

[3] http://www.openstreetmap.org/

Figure 3: *The combination of terrain structuring and visibility optimization with progressive objects and textures allows for a continuous adaptive navigation experience in planet-sized virtual worlds, with visual features ranging from overall country limits to detailed, geolocalized street furniture. In this example data are extracted from OpenStreetMap*[3] *(127GB database in March 2010) and converted into X3D Earth representation. The buildings are generated using adaptive procedural reconstruction on client-side using the building footprints contained in the OSM database.*

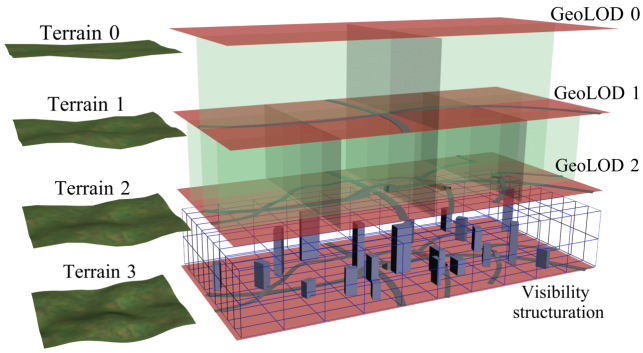

Figure 4: *The altitude of the viewpoint drives the management of the levels of detail: at large distances we use the file-based levels of detail provided by the* `GeoLOD` *nodes of X3D. At groundlevel we use a more optimized representation of the world based on visibility relationships between cells of a uniform grid. The terrain itself is managed separately using dedicated streaming techniques based on HTTP/1.1 chunk reads.*

The navigation within the cell structure is made possible by introducing a new sensor node, called `CellTracker` (Figure 5). Given the URLs of the entry gates of the structure, this node tracks the viewpoint and communicates with the current cell to fetch the visible cells in the neighborhood. The visibility streaming mechanism is completed with data prefetching and cache management techniques: an oracle determines and fetches the next potentially visited cells using viewpoint tracking and motion prediction. Conversely, formerly visible cells are discarded based on a distance criterion computed on the adjacency graph of the cells.

```
CellTracker  : X3DSensorNode {
    MFString [in,out] entryCells [""]
}
```

Figure 5: *Prototype of the* `CellTracker` *node used to track the current location of the user with respect to the cell-based structure.*

The transition from the `GeoLOD`-based structure to the visibility-based representation occurs when navigating through the leaves of the `GeoLOD` hierarchy. This leaf level contains the list of adjacent cells, allowing the system to update the *CellTracker* node and quickly determine the entry point within the visibility-based structure. This determination can be further optimized if the cells define a uniform, axis-aligned grid. Once the set of potentially visible objects has been determined, our system adaptively fetches and renders the corresponding geometric and photometric structures.

3.2 Adaptive Objects and Textures Streams

Even when the set of potentially visible objects is determined, loading such objects in full resolution may impact the interactivity of the world and potentially consume prohibitive amounts of memory. To avoid such issues, we determine overall budgets for network bandwidth, texture memory and polygon count. The computation of those budgets relies on two principles: real-time monitoring of key performance indicators and statically defined configuration parameters. The former considers the current per-frame render time, the number of visible polygons and the amount of graphics memory used for texture storage. The latter is a set of user-defined values such as a target frame rate or the maximum amount of memory dedicated to textures. Results and the budget management are illustrated in the accompanying video.

Once the overall budgets for bandwidth, polygon count and texture memory have been allocated we split this budget over the set of potentially visible objects. The budget share allocated to each object is defined by an estimate of its current visual importance, typically using pixel coverage of object projections (easily performed using occlusion queries on graphics hardware). Depending on the variations of its allocated budget, a node may trigger unloading of excess data, request the server to provide additional levels of details, and update its internal structures.

In this approach many nodes may simultaneously issue requests for additional geometric or texture details. Using HTTP/1.0 [Berners-Lee et al. 1996], each request would have to wait for the previous request to be fulfilled before being issued. The only alternative would be to establish a separate connection for each request, which would also introduce much delay in the responses. HTTP/1.1 introduces support of persistent connection, chunk reads and *pipelining*, in which multiple requests can be issued successively without waiting for responses. These mecanisms allow each node to issue requests without any overhead or stalls.

Building upon those characteristics we use methods for adaptive streaming of multi-resolution meshes and textures. In each case we represent the various levels of details using contiguous chunks of data in a single file. Each file starts with a first level of detail along with metadata providing the location of the next levels in the file. Figure 6 illustrates this organization in the case of a binary tree in which the metadata indicate the offsets between the current node and each of its children.

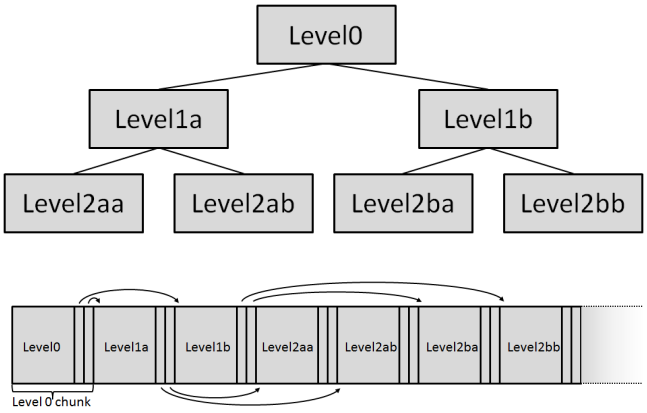

Figure 6: *Representation of a binary tree compliant with HTTP/1.1 chunk reads: each node stores its data as well as information regarding file offsets towards each of its children.*

3.2.1 Multi-resolution meshes

Many applications use several levels of detail to represent a given object for performance purposes: if an object has a small visual importance, the system uses a low-resolution representation of the object. Conversely, visually important objects benefit from higher quality representations. This technique, known as *discrete levels of detail*, is typically supported by the `LOD` node of the X3D specification: based on a set of distance thresholds associated to the corresponding independent object representations, the system chooses the representation adapted to the current distance between the viewpoint and the user. This technique effectively reduces the polygon count for visually unimportant objects, hence improving the overall rendering performance. However, due to the lack of specific streaming approaches, the `LOD` node requires transmitting all the levels of detail prior to rendering. This problem can be addressed using a combination of `Inline` nodes and specific scripts, at the expense of computational efficiency. Also, those levels are stored in independent files: a given object is then represented multiple times, resulting in bandwidth waste due to redundancies.

On the rendering side, let us consider an object advancing towards the viewer. In this case, each time the object passes a distance threshold, the `LOD` node switches the object representation to a higher level. If difference of polygon count between two successive levels is too high, the level switch becomes visible, yielding annoying *popping* artifacts. This problem can be overcome using more levels of detail, resulting in higher visual quality at the cost of memory and bandwidth consumption.

Instead, in the spirit of [Hoppe 1996], we use a continuous technique: a single representation of the geometry of the objects is encoded to provide continuous levels of details and progressive transmission capabilities. Object meshes can be progressively refined with a per-vertex granularity, reducing the popping artifacts to a minimum. The file layout for a progressive mesh (Figure 7) follows the global structure described above, in which each vertex split operation is considered as a separate level of detail. The structure then contains as many levels of detail as available split/merge rules. Also, the size of such rules is constant, avoiding the need for storing offsets towards the next level of detail.

While slightly different from the generic structure, this layout is particularly adapted to the dynamic allocation of polygon and bandwidth/latency budgets: if an object already contains n faces and is allowed $n+m$ for the next frame, the system simply issues a chunk request for m times the size of a split rule. Note that the adaptation is performed not only on the polygon budget, but also on the availability of the network link: if the polygon budget recommends adding m faces while the bandwidth budget allows for $m-k$ faces, then only $m-k$ faces are downloaded and added to the object.

When rules are downloaded and applied to the mesh on client-side, the file pointer gets advanced to the next available rule. Conversely, if the budget for an object is reduced, the geometry is degraded and the pointer is decremented by the number of discarded rules. The photometry of the objects is then adjusted using multi-resolution textures.

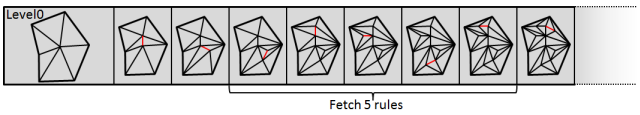

Figure 7: *A progressive mesh is represented by its base representation, followed by a sequence of fixed size split/merge rules. The addition of n vertices is simply performed by requesting the server to send a file chunk of n times the size of a rule, starting from the position of the last chunk request.*

3.2.2 Multi-resolution textures

Many approaches including [Shapiro 1993; Said and Pearlman 1996; Hewlett-Packard 1997; Marvie and Bouatouch 2003] have been devised for efficient multi-resolution representation and streaming of texture maps. Among them, zerotree based techniques [Shapiro 1993] such as JPEG2000 [ISO/IEC FCD 2004] feature scalable progressive transmission based on resolution or PSNR. However, this resulting compacity comes at the expense of computational efficiency for decoding.

Building upon the ideas of low-redundancy and chunk reads, our approach is based on the tiled pyramidal format described in [Mar-

vie and Bouatouch 2003]. This format can be seen as a progressive PNG format dedicated to texture maps. It provides an efficient representation for progressive texture maps allowing for dynamic loads and unloads of mipmap levels. As shown in Figure 8, this file format first stores a base dataset representing the texture map. For example, resolutions from 1×1 to 32×32 can be stored in this first file chunk. The higher resolution is represented in the next chunk as follows: Each texel of the lower resolution is subdivided into 4 subtexels. The values of 3 subtexels are explicitly stored, while the fourth is reconstructed using the values its neighbors and the lower resolution texel, as well as a small residual value compensating the error induced by integer arithmetics. A given mip level is then the set of additional subtexels plus the residual values. Further lossless compression is achieved using per-level ZIP compression.

Our approach is however not limited to [Marvie and Bouatouch 2003]: the same principle can be applied using classical formats such as interlaced GIF (Figure 9) or PNG. Also, alternative representations such as core JPEG2000 could also be slightly adapted to fit this chunked-based streaming scheme [Deshpande and Zeng 2001].

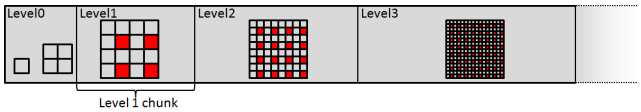

Figure 8: *Progressive textures are encoded by chunks representing partial mip levels, and losslessly reconstructing the missing texels (in red) on client-side.*

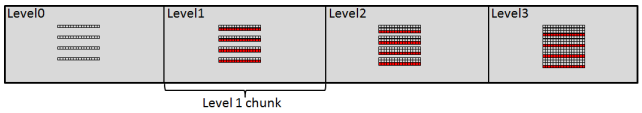

Figure 9: *Streaming can also be performed using existing progressive image file format, such as interlaced GIF. This format contains four levels of details: the first level contains the first line of each four-row blocks of the image, the second level the second line etc. Each level can be downloaded as a file chunk through HTTP/1.1.*

In our context the advantages of [Marvie and Bouatouch 2003] are multiple: first, the representation corresponds directly to the mipmaps structure of graphics hardware. Consequently, this approach avoids the need for generating the mip levels, relieving the CPU or GPU from the filtering task. Also, the reconstruction of the texels is performed using simple bitwise operations, particularly suited to low-end terminals. In counterpart, JPEG2000 would provide better compression at the transmission level at the expense of decompression complexity, especially for regenerating mipmap levels. However, note that this complexity can be reduced using index tables embedded within the JPEG2000 files.

At render time, the memory budget allocated to each texture is determined by the system based on the overall number of visible texture maps within the set of potentially visible objects, as well as visual importance cues such as the coverage hints of the texture-mapped objects. If the currently available resolution is not sufficient, the system issues a file chunk request to obtain the additional information required to build the next resolution level.

3.2.3 Multi-resolution terrains

The geospatial component of X3D provides a way of representing terrain elements using discrete levels of detail stored in separate files. While this approach proves effective, successive levels of detail contain redundant data, resulting in a waste of bandwidth. Many alternatives have been proposed to address redundancy issues, such as [Hoppe 1998; Cignoni et al. 2003; Losasso and Hoppe 2004; Pouderoux and Marvie 2005; Lerbour et al. 2009; Lerbour et al. 2010]. Our system implements the approaches by [Lerbour et al. 2009] to meet our needs in terms of progressivity, computational efficiency and ease of transmission. As described above in the context of progressive textures, the main idea is the addition of data on top of lower-resolution levels to progressively reconstruct the full representation of the terrain.

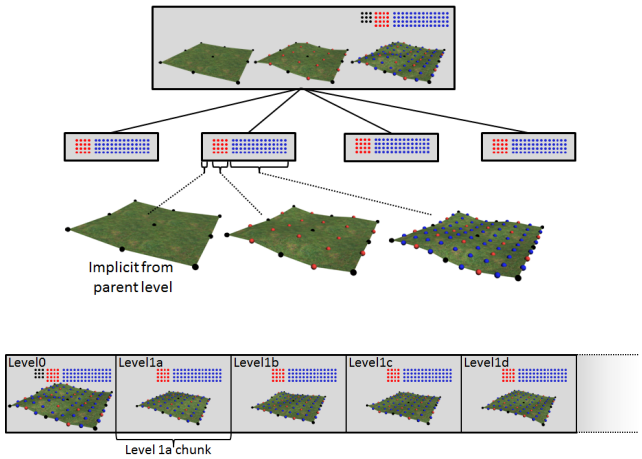

Figure 10: *Each level of the terrain representation contains three non-redundant levels of detail. The first inner LOD (black dots) is simply the highest inner LOD of the previous level, and is thus not explicitly stored for levels higher than 0.*

The elevation of the terrain is encoded within a tree data structure, for example a quadtree as illustrated by Figure 10. Each node of the tree contains three complementary levels of detail. The first LOD of inner nodes is implicitly provided by the last level of their parents to avoid redundancy. Each node of the tree is stored within chunks of the main terrain file on the server, and fetched on-demand by the client. This technique provides the user with an immediate access to the terrain, whose representation gets refined during navigation using simple chunk reads.

The rendering of the terrain also complies with the dynamic budget allocation: the tree is refined to meet the constraints induced by the polygon and bandwidth/latency budgets. Depending on the application, the budget allocated to each node can be based on several criteria such as the distance to the viewpoint, the screen projection of the node or the surface roughness. The adaptivity of the photometry of the terrain is ensured by maintaining a similar tree containing texture information [Lerbour et al. 2010].

3.3 Procedural models

The techniques described above aim at representing and streaming progressive meshes and texture maps. While compact and easily streamable, further performance can be achieved for specific objects. For example, many buildings can be represented by a set of construction rules, or grammar, and reconstructed at any level of detail. Such grammars tend to be very small in terms of storage, and a single grammar can be reused to generate infinite variations of a given building type. This technique known as *procedural modeling* is often used at the design stage to create large environments such as cityscapes, which are then stored as regular meshes for streaming

and rendering.

In [Marvie et al. 2003], the authors avoid the need for explicit geometry generation prior to transmission: the full grammar is first transmitted to the client using HTTP file read. Then, depending on available resources of the client, the grammar is developed dynamically to generate the appropriate geometry level (Figure 1(e)). This technique called *procedural LOD control* allows for drastic bandwidth savings and implicitly adaptive geometry, proving particularly useful for the navigation in complex urban environments.

3.4 Servers Localization

The world delivery scheme described in this paper is fully adaptive to the client capabilities and the network bandwidth. However, as any network-based solution, our approach benefits from low latencies and high upload bandwidth for the servers. Therefore, the system will perform best using replicated data centers spread in multiple locations of the real world, and chosen depending on the current location of the clients. Note that the servers only host the world resources, ignoring the presence and synchronization of the users. Consequently, the replicated servers need to be synchronized only if the world resources are changed, which is usually a non-intensive task. The interaction of the users is then performed using a decentralized approach.

4 World Synchronization

4.1 Exisiting Technologies and Related Standards

The challenge of massive multi-user synchronization mostly consists in finding an optimal balance between heterogenous network latencies and bandwidths, capabilities of the client hardware, as well as the level of interaction of each users. Most existing synchronisation techniques are either fully centralized or fully decentralized.

Centralized architectures are generally built upon a client/server model. Clients only communicate with a server which, in turn, propagates world changes or connection updates towards the other clients. Those architectures are widely spread in commercial Massive Multi-User Online Games (MMOG) such Second Life [Stenio et al. 2007] or World of Warcraft, and are usually based upon the `TCP` protocol. The centralization is particularly desirable for hosting commercial games: the state of the world can be easily monitored, allowing the game producer to easily ensure the quality of service and retain user logins for billing purposes. While user synchronisation is straightforward in this context, the major bottleneck comes from the network interface of the server. As TCP generates a lot of traffic, the saturation of the servers may result in denial of service or high latency issues [Svoboda et al. 2007]. Another important aspect relates to the persistence of the world: if the server crashes, the entire world becomes unavailable. To overcome these problems the servers have to be replicated and constantly synchronized to increase the available bandwidth and reduce the crash probability.

Decentralized architectures appear more robust and cost-effective due to the lack of dedicated servers: the bandwidth is distributed among all the clients, and the probability of a simultaneous crash tends to be negligible. However, synchronizing a decentralized world is far from trivial. Many architectures and synchronization strategies have been proposed to prevent saturating the network links of the clients. First research works have been performed in the early 80s within the SIMNET [Miller and Thorpe 1995] for distributed combat simulation. This work has been later extended and standardized in the DIS protocols [IEEE-Std-1278 1996] and implemented in NPSNET [Zyda and Pratt 1991]. HLA standard protocols [IEEE-Std-1516 2010] extend DIS by introducing cell partitioning of virtual environments and multicast groups. Strategies and architectures for multicast grouping have been proposed in SCORE [Léty et al. 2004], NPSNET [Zyda and Pratt 1991] or DIVE [Carlsson and Hagsand 1993]. However, the use of multicast addresses is generally restricted or prohibited. Alternative protocols based on peer-to-peer approaches have been proposed in VON[Hu et al. 2006] and implemented in VAST.

We propose a solution based on X3D for easy network communication and synchronization in the context of centralized, decentralized and hybrid multi-user virtual worlds.

4.2 Towards a Scalable Synchronisation of Online Worlds

The X3D standard provides a high level networking component based on DIS protocols which, historically, target military applications. More generic alternatives have been proposed [Bouras et al. 2005] but are limited to centralized architectures and require specific server software. Our hybrid approach avoids the need for such servers while preserving easy synchronization.

We introduce a novel X3D node called `NetSync` for low level peer-to-peer network communication. This node provides an easy way to connect and synchronize a list of known peers and share data among them. The genericity of `NetSync` combined with scripting and user-defined `PROTO` nodes allows for easy design of any kind of centralized, decentralized or hybrid network architecture.

4.2.1 The `NetSync` Node

```
NetSync {
  SFBool   [in,out] authorizeOtherPeers FALSE
  MFString [in,out] peers               []
  MFString [out]    peerDisconnect      []
  SFString [in,out] protocol            "HTTP/1.1"
  SFString [in,out] port                "80"
  SFBool   [in,out] enablePipelining     TRUE

  # list of additional user [in] or [out] fields

  SFFloat [in]  send_velocity
  SFVec3f [in]  send_position
  SFVec3f [out] position_received
  ...
}
```

Figure 11: *Prototype of the `NetSync` node*

The `NetSync` node (Figure 11) is a versatile communication node for easy connection to multiple peers. Centralized architectures can be implemented by instantiating a `NetSync` node in the scene graph opened in a X3D browser on the server computer (Figure 12). The `peers` field of this node potentially stores the list of clients which are allowed to connect. The scene graph browsed by the clients also contains a `NetSync` node *with the same name*, whose `peers` field only contain the address of the server.

Authorizations and user management issues are addressed using the `authorizeOtherPeers` flag. By default, the `NetSync` node rejects connections issued by peers which are not declared in the `peers` field. If the flag is set to `TRUE` anyone can connect the `NetSync` node regardless of the contents of the `peers` field. The connection between peers thus only require the knowledge of their respective IP addresses and an agreement on a common name for the `NetSync` nodes. Peer disconnection is handled using the `peerDisconnect` output field.

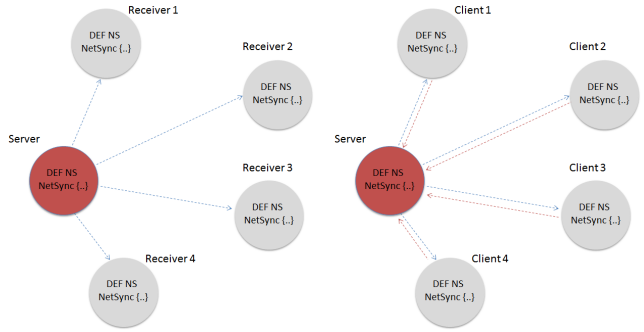

Figure 12: *Unidirectional broadcast (left) and client/server (right) network architectures using* `NetSync` *nodes.*

The communication between peers is performed using dynamic user-defined fields in the spirit of `PROTO` or `ComposedShader` nodes. Depending on the application, those fields may include properties such as the velocity of the user or his current position in the virtual world. Attention must be paid to the field names, which must be exactly the same for each of the `NetSync` nodes. Also, as mentioned above, the names of connected `NetSync` nodes must be identical: this allows a single scene graph to contain multiple `NetSync` nodes dedicated to various tasks.

The `NetSync` node supports the UDP, TCP and HTTP/1.1 protocols depending on the application. For example, professional applications would tend to prefer TCP for reliability reasons, at the expense of the bandwidth. The simplicity of UDP makes this protocol particularly useful for high-frequency synchronization and massive multi-user worlds, in which the precision of the interaction is not critical. As TCP and UDP may be faced to deployment issues regarding the communication on specific ports through proxies and firewalls, HTTP/1.1 can be used for low-frequency, generic synchronization. Depending on the current protocol, the `enablePipelining` field activates pipelining on HTTP/1.1, TCP-no-delay on TCP and has no effect in UDP.

In the framework of streaming and synchronization of multi-user worlds, we used the `NetSync` node to implement a hybrid synchronization architecture based on peer-to-peer connections combined to a yellow pages server as peer neighbor discovery service.

4.2.2 P2P User Synchronization

Multi-user online worlds usually represent users using avatars associated to a set of possible actions such joining the world, walking or sending messages. High frequency synchronization of the users is performed by a hybrid network architecture implemented using `NetSync` nodes (Figure 13) along with specific fields for event spreading.

The first step for implementing our approach consists in choosing a super peer which will host a list of all the peers connected to the virtual world. This system, known as a yellow pages server, allows newcomers to retrieve the peer list for later interaction. Once the list is retrieved, a `NetSync` node is dynamically created on each client and connected to the peers. This node contains a number of specific fields such as the location of the corresponding avatar and potential chat messages (Figure 14). At the end of a session, the disconnecting peer notifies the yellow pages server and destroys its synchronization node.

An important aspect of our hybrid peer management based on yellow pages servers lies in its ease of use and its robustness: similar to a centralized approach, the yellow pages server provides a simple way of identifying the peers. However, each peer retrieves and

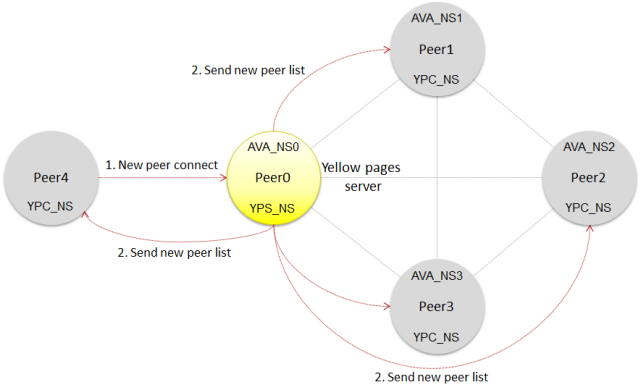

Figure 13: *Hybrid synchronization architecture based on a yellow pages server using the* `NetSync` *node combined with X3D* `PROTO` *and scripting facilities. A* `PROTO` *encapsulating a yellow pages server is instanced by a super peer. Newcomers simply instance a yellow pages client (YPC_NS) to contact the server and retrieve the list of connected peers. Each client then instances an avatar* `NetSync` *node (AVA_NS0..N) connected to all the neighboring peers for multi-user synchronization.*

stores the entire list of peers: if the yellow pages server gets disconnected another peer can seamlessly take this role without any unwanted disconnections or inconsistencies. Finally, in the context of massive multi-user worlds, a single yellow pages server may be overwhelmed by the requests from the peers. In this case several servers may be chosen: those super peers would then instantiate appropriate `NetSync` nodes for synchronizing their peer lists.

As in the real world, the interaction between avatars is only possible within close neighborhood in the virtual world. Based on this observation, further efficiency can be obtained by network zoning: the peers can be clustered dynamically based on their relative locations of the corresponding avatars. This approach spreads the load on numerous yellow pages servers and reduce the number of simultaneous connections opened by the peers.

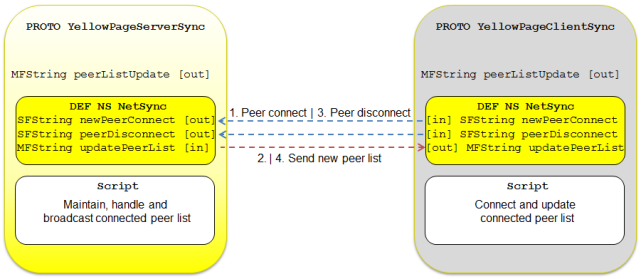

Figure 14: *Yellow page client/server implementation details based on the new* `NetSync` *node extension and X3D prototyping/scripting facilities.*

5 Results

We implemented our approaches within our X3D browser and applied for multi-user interaction in a virtual world featuring detailed geometry and textures (Figure 15). Depending on the location of the avatars in the virtual world, only the potentially visible objects are streamed to the client, according to the current budgets and framerate constraints (Figure 16, top). The synchronization with the other users generates a small network traffic as shown in the bottom of Figure 16. In this world the synchronization is perform using the TCP protocol.

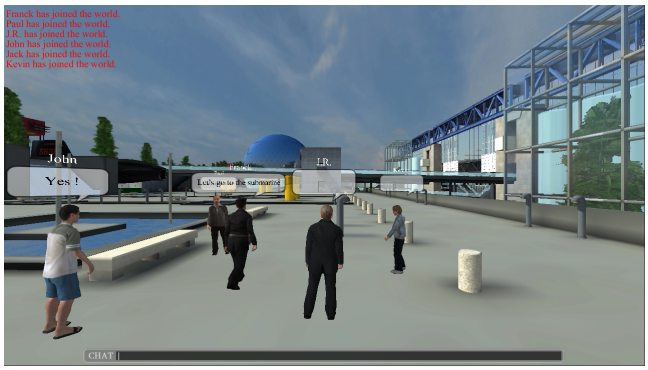

Figure 15: *Multi-user interaction in a streamed virtual world based on adaptive streaming of geometry and textures combined with peer-to-peer synchronization.*

As shown in the accompanying video, a first peer takes the role of yellow pages server. Then, newcomers connect this server to register and obtain the list of other connected peers. Avatar motions, interactions and chat messages are then synchronized in real-time.

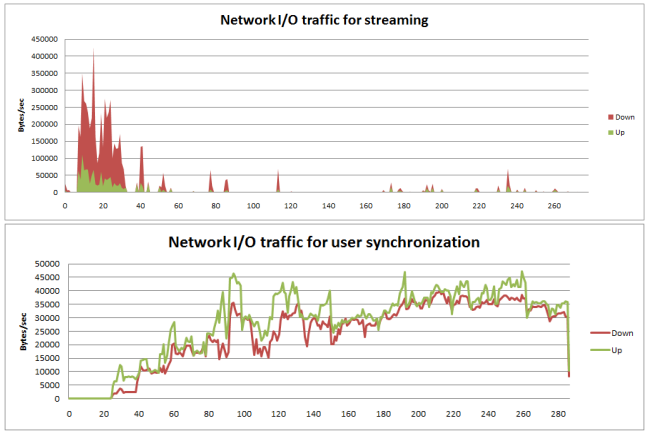

Figure 16: *Traffic analysis for streaming and synchronization during a multi-user session. Depending on the movements and speed of the user the visible contents of the scene are requested by the clients and adaptively streamed from the server. If the set of potentially visible cells and objects is constant, no further data are streamed (seconds 120 to 160). When users join the world, interaction data are exchanged dynamically to ensure peer-to-peer synchronization.*

6 Conclusion

The multiplication of collaborative networking and 3D-enabled devices in our everyday life enforces a need for easily deployable multi-user virtual worlds, regardless of hardware capabilities or network configuration and bandwidth. We propose a unified approach for adaptive streaming of planet-size multi-user virtual worlds, providing the user with a click-and-play experience. Based on the capabilities of HTTP/1.1, our system is based on passive servers and does not require any dedicated protocol, server software or specific proxy configuration.

Our adaptive engine focuses on the streaming of geometry and textures on-demand depending on live estimates of the quality of network links and of available resources on the client. The resources are efficiently streamed from geographically close servers using packed binary representations.

The synchronization of multiple users is performed using a localized peer-to-peer approach, in which the clients are connected with respect to their relative locations in the virtual world. Even though we designed a specific X3D node for network synchronization, similar nodes are also available in other X3D browsers. The streaming and synchronization aspects can thus be supported in any X3D browser at the cost of minor extensions. This provides the user with instant access to massive virtual worlds on any device while relieving the world owners from the management of specific software and multiple target platforms.

References

BERNERS-LEE, T., FIELDING, R., AND FRYSTYK, H., 1996. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945.

BOURAS, C., PANAGOPOULOS, A., AND TSIATSOS, T. 2005. Advances in x3d multi-user virtual environments. In *Proceedings of IEEE International Symposium on Multimedia*.

CARLSSON, C., AND HAGSAND, O. 1993. DIVE a multi-user virtual reality system. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, 394 –400.

CAVAGNA, R., BOUVILLE, C., AND ROYAN, J. 2006. P2P network for very large virtual environments. In *Proceedings of VRST*.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3, 505–514.

DESHPANDE, S., AND ZENG, W. 2001. Scalable streaming of JPEG2000 images using hypertext transfer protocol. In *ACM Multimedia*, 372–381.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T., 1999. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616.

HEWLETT-PACKARD. 1997. Internet imaging protocol v1.0.5.

HOPPE, H. 1996. Progressive Meshes. In *Proceedings of SIGGRAPH*, 99–108.

HOPPE, H. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings of the IEEE Conference on Visualization*, 35–42.

HU, S.-Y., CHEN, J.-F., AND CHEN, T.-H. 2006. VON: a scalable peer-to-peer network for virtual environments. *IEEE Network 20*, 4, 22 –31.

IEEE-STD-1278. 1996. IEEE standard for distributed interactive simulation - application protocols. *IEEE Std 1278.1-1995*.

IEEE-STD-1516. 2010. IEEE standard for modeling and simulation: High level architecture. *IEEE Std 1516-2010* (18), 1 –38.

ISO/IEC FCD. 2004. 15444-1: JPEG2000 image coding system, part 1: Core coding system.

JEAN-EUDES MARVIE, JULIEN PERRET, K. B. 2005. The FL-system: A functional L-system for procedural geometric modeling. In *The Visual Computer*.

LERBOUR, R., MARVIE, J.-E., AND GAUTRON, P. 2009. Adaptive streaming and rendering of large terrains: A generic solution. In *Proceedings of WSCG*.

LERBOUR, R., MARVIE, J.-E., AND GAUTRON, P. 2010. Adaptive real-time rendering of planetary terrains. In *Proceedings of WSCG*.

LÉTY, E., TURLETTI, T., AND BACCELLI, F. 2004. SCORE: a scalable communication protocol for large-scale virtual environments. *IEEE/ACM Transactions on Networking 12*, 247–260.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics 23*, 3, 769–776.

MARVIE, J. E., AND BOUATOUCH, K. 2003. Remote rendering of massively textured 3D scenes through progressive texture maps. In *Proceedings of IASTED*, vol. 2, 756–761.

MARVIE, J.-E., AND BOUATOUCH, K. 2004. A VRML97-X3D extension for massive scenery management in virtual worlds. In *Proceedings Web3D*, 145–153.

MARVIE, J. E., PERRET, J., AND BOUATOUCH, K. 2003. Remote interactive walkthrough of city models. In *Proceedings of Pacific Graphics*, vol. 2, 389–393.

MILLER, D., AND THORPE, J. 1995. SIMNET: the advent of simulator networking. *Proceedings of the IEEE 83*, 8, 1114 – 1123.

POUDEROUX, J., AND MARVIE, J.-E. 2005. Adaptive streaming and rendering of large terrains using strip masks. In *Proceedings of ACM GRAPHITE*, 299–306.

ROYAN, J., GIOIA, P., CAVAGNA, R., AND BOUVILLE, C. 2007. Network-based visualization of 3D landscapes and city models. *IEEE Computer Graphics & Applications 27*, 6, 70–79.

SAID, A., AND PEARLMAN, W. 1996. A new fast and efficient image codec based on set partitionning in hierarchical trees. *IEEE Trans. Circuits and Systems for Video Technology 6*, 3, 243–250.

SHAPIRO, J. 1993. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. Signal Processing 41*, 12, 3445–3462.

STENIO, F., CARLOS, K., DJAMEL, S., JOSILENE, M., AND RAFAEL, A. 2007. Traffic analysis beyond this world: the case of second life. In *Proceedings of Nossdav*.

SVOBODA, P., KARNER, W., AND RUPP, M. 2007. Traffic analysis and modeling for world of warcraft. In *Proceedings of IEEE International Conference on Communications*, 1612 –1617.

WEB3D.CONSORTIUM. 2008. Information technology computer graphics and image processing extensible 3d (x3d). *ISO/IEC 19775-1:2008*.

ZYDA, M., AND PRATT, D. 1991. NPSNET: A 3d visual simulator for virtual world exploration and experimentation. In *Proceedings of SID International Symposium*, vol. 22, 361–364.