Managing dynamic entities in mobile, urban virtual environments

Antti Nurminen
antti.nurminen@hut.fi
Helsinki University of Technology

ABSTRACT

Mobile networked virtual environments (mNVE's) are a new, emerging type of virtual environments. *Mobile 3D maps* that support dynamic entities and communication between clients are a subcategory of mNVE's, intended for navigation and location-based information browsing. Models and entities portrayed in 3D maps represent real environments and entities, such as buildings, vehicles and people. Our main contribution is in developing a lightweight and scalable scheme for real dynamic entity management and visibility culling by exploiting geometry of urban environments, the *honesty* of locally positioned clients and the lack of interference between clients. We bind moving entities to a topological network consisting of street segments, crossings and larger areas, all associated to precalculated visibility cells. Our system reduces visibility determination to a simple cell occupation logic, performed at smart clients or proxies. In this scheme, servers act as fast message passing switches, managing client subscription and query tables, simply forwarding state update messages. Computational scalability is ensured by transferring computations to client side, and networking scalability by spatially localized servers, which allow roaming by subscribing to each others' neighboring visibility cells.

Keywords: 3D maps, mobile networked virtual environments, 3D user interfaces

1 INTRODUCTION AND RELATED WORK

Mobile 3D graphics API's, mobile 3D hardware and cellular networks have reached the point where implementations of advanced, networked and graphically rich applications are possible on mobile devices. Despite these developments, mobile devices are still *thin*, and cannot directly present large and detailed, dynamic worlds. We attack this challenging optimization problem and develop a scalable mobile platform for visualizing static and dynamic objects in urban environments with near real time tracked real world entities, rendering the scene at interactive refresh rates, and in a realistic manner.

Our work has connections to networked virtual environments and computer graphics optimizations in general. We discuss previous work, and exploit the features of our environment to create a lightweight and scalable solution, based on precalculated cell-to-object and cell-to-cell visibilities, topological data structures and client-side logic for positioning decisions and distributing position updates.

1.1 Mobile maps

Maps are representations of real environments. The level of abstraction may vary, from symbolic 2D representation to realistic 3D. Most commercial mobile maps, such as TomTom [Tom06], have been designed for navigational purposes, and may feature static location-based information, such as restaurants, museums, and other points of interest. 2D map views are based on static raster pictures, or real-time rendered vector graphics. A currently popular view mode in car navigation systems is the *perspective 2D view*, portraying 2D street networks from the street level with a perspective transformation. Mobile map research projects have yielded prototypes with various navigational features and interaction methods, supporting multimedia and online searches [Che00, Pos02, Bau01].

The key idea in 3D maps, in contrast to 2D maps, is the direct recognizability of the environment - when rendered in full 3D, including buildings and all other features of the environment, the virtual scene should match the real world, facilitating unambiguous navigation.

The first attempts at creating mobile, interactive 3D maps faced severe technical limitations. Without 3D hardware, and without optimizations, for example the *3D City Info* project attempted to use a realistic VRML city model, but had to perform the first field experiments with pre-rendered images on web pages [Rak01]. The *TellMaris* project applied simple spatial culling, and was able to render low resolution textured models at interactive rates [Prz05], but without routing or online search capabilities. With visibility information embedded in VRML models, and low resolution textures, Burigat and Chittaro [Bur05] achieved 4-5fps for a city square model, with information content retrievable from the model. These prototypes did not support progressive model downloading, nor dynamic entities.

When dynamic entities and message passing between clients are introduced to mobile 3D maps, they can be viewed as a subset of mobile networked environments (mNVE's), where the 3D content represents real environments, and the entities represent real people and vehicles. In this sense, such mobile 3D maps are *real virtual environments*. Figure 1 presents our case, an urban environment with tracked, dynamic entities.

1.2 Networked virtual environments

Networked virtual environments (NVE's) are simulated worlds, where multiple users can interact with the shared environment, and each other. NVE's include text-based games such as *multi-user dungeons* (MUDs), teleoperation applications, massive military combat simulators and immersive, shared environments such as *collaborative virtual environments* (CVE). First NVE's were military simulators, such as the SIMNET [Joh87], but academic use and entertainment industry soon adapted similar technologies. The most popular NVE's are currently network games, such as the massive multiplayer online role-playing game (MMORPG) *World of Warcraft* (WoW) or the first-person shooter (FPS) *Counter-Strike*, with millions of users.

Most NVE's include a 3D front-end. Large, increasingly more realistic environments easily exceed the capabilities of any given 3D hardware. Smart algorithms are required to reach interactive rendering speeds. The two fundamental methods are *visibility culling* and *level of detail* (LOD). Visibility determination methods remove those objects from the rendering pipeline that are not visible in the current view, and LOD techniques use screen space metrics to determine how accurately objects need to be rendered. LOD methods include optimization of both geometry and surface detail (shaders, textures).

Networking requirements of NVE's depend on the application. For example, a networked chess would only involve two players, whereas a military simulation or a MMORPG might need to scale up to hundreds of thousands of units or players. In a client-server model, even a chess server could be exhausted, if it was required to serve a million concurrent clients. As the games are independent, scalability can be addressed by simply increasing the number of servers. If the world allows users to interact, as in military simulations or games, message passing must be facilitated, and independent servers are not sufficient.

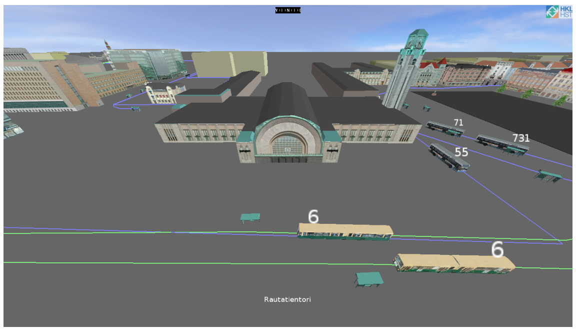

Figure 1: Real dynamic entities: GPS tracked public transportation

Interest management [RMa95] and *communication visibility* [Cap97] techniques have been developed to minimize network traffic between clients. Common solutions utilize various area-of-interest (AOI) based approaches and direct visibility, but the related computation and decision making is performed at server side, causing a potential computational bottleneck. In the *Player/Ghost* model, transmissions are optimized by *honest* Players, which send updates only when a locally computed extrapolated state deviates from the real one significantly [Bla92].

1.3 Network topologies

Network topology has a significant impact on the potential scalability of NVE's, both computationally and network-wise. Client-server solutions often require the server to maintain a simulation of the entire world, and typically server-to-client network traffic rises linearly with the number of clients. For example, one of the most popular networked first person shooter games, *Counter-Strike*, was observed to follow this rule very accurately [Far04]. Client-server solutions are considered good for event- and behavior-rich environments, where *persistency* and *consistency* are important, but the server and the network easily become bottlenecks. Common client-server FPS games allow only 32–64 simultaneous users per server.

Peer-to-peer (P2P) network based systems could scale infinitely, if each client would perform only a small part of the whole world simulation, and transmit its data only to a limited number of other clients. In the NPSNET-IV military simulator, clients distributed their states to every participating client, and a maximum of 300 Players was reached on a 10Mbit/s ethernet [RMa95].

In a P2P system, every participating unit must be honest. In addition, resolving parallel actions, where client states diverge, is difficult [Mar06]. P2P networks suit the situation where divergence of client states is unlikely, and which require computational scalability.

2 VISIBILITY OF STATIC SCENES

Complex models can be rendered at interactive rates even in lightweight systems, given that the complexity is reduced by rendering only the actually visible parts of the model, using appropriate levels of details. Naturally, the run time computations required to achieve such an *output-sensitive* situation should be minimal. The classical scenario has been a walkthrough of a static, densely occluded scene, a situation usually found indoors. [Coh03] provides a comprehensive survey of such methods. In these applications, the world is typically divided into a hierarchical structure, and various culling techniques applied to select the parts of the scene currently in view.

Most visibility algorithms aim for *conservativeness*, where the method ensures that all visible geometry is rendered, with the risk of including some occluded geometry. The possible degree of *aggressivity* in visibility determination depends on the application, and the situation. In practice, even an *approximate* solution may provide sufficient quality.

2.1 Spatial subdivision

Visibility algorithms are tied to the underlying spatial subdivision algorithm. For static indoor scenes, the binary space partition algorithm (BSP) [Fuc80] has been popular. A BSP tree structure can directly represent a correctly created B-rep model, avoiding any external data structures, and can also be used for fast collision detection [Ar00].

Octree based spatial subdivision algorithms divide the space into a hierarchy of volumes. 2D spatial data can be divided to a hierarchy of quads using *quadtrees*. Octrees and quadtrees usually serve as separate, assisting data structures. They provide good localization, and the hierarchical structure suits various culling schemes, such as view frustum culling.

The *hierarchical Z-buffer* algorithm[Gre93] utilizes both an object-space octree, and image-space Z buffer hierarchy. The *hierarchical occlusion map* (HOM) stores opacity and occluder distance information separately [Zha98], creating potential occluders in a preprocess.

2.2 Potentially visible sets

The concept of potentially visible sets (PVS) was developed in the seminal work by Airey [Air90] and Teller [Tel92]. In this scheme, the world is divided to *cells*, connected to each other through *portals*, openings between the cells such as doorways. Cell-to-cell or cell-to-object (or cell-to-polygon) visibilities are precomputed, and at run time, the objects deemed visible from the current cell are rendered. [Tel92] also defines *detail objects*, which are discarded as non-occluding, small objects. Later research improves upon this work. For example, [And00] defines the *hardly visible set*, objects that contribute only little to the scene and can be discarded. In city scenarios, [Cap97] classifies visibility into *graduated visibility sets*, for objects of varying visibilities. The visibility precomputation is also associated with the required level of detail of the models: the *vLOD* system [Chh05] binds these aspects together.

Potentially visible sets are a very powerful tool for visibility culling. Lookup functions have minimal overhead in fetching precomputed visibilities, given sufficiently simple visibility list compression algorithms. With PVS systems, there is no need for expensive computations or view dependent scene structure rearrangements, such as BSP tree reconfigurations.

For our case, with free viepoint, we subdivide our space to 3D voxels (view cells), and apply a cell-to-object PVS algorithm, using façades and roofs as objects. A precomputation stage creates visibility lists and compresses them into difference clusters. Our static object culling scheme is described in [Nur06].

3 VISIBILITY OF DYNAMIC OBJECTS

Dynamic objects are not part of the static world, and their visibilities cannot be simply preprocessed. [Chr92] asserts three requirements for an algorithm supporting dynamic 3D scenes, the abilities to

1. Change the camera view
2. Add objects to the scene
3. Delete objects from the scene

A dynamic entity is then managed by deleting it from the old position, transforming it, and inserting back to the scene. This introduces significant overhead, a problem long recognized. In the context of managing dynamic objects within BSP based urban scenes, Fuchs suggested to divide static and dynamic objects to separate BSP trees, where the static BST trees would not intersect the paths of dynamic objects [Fuc83].

In addition to the insertion overhead, visibility would need to be determined each frame. Sudarsky and Gotsman offer a possible relief: each moving object is replaced by a *temporary bounding volume* (TBV), which contains the object during a *validity period* [Sud97]. During this time, run time visibility calculations rely on the TBV. A TBV is created based on a priori knowledge of the object's behavior. For objects with well known trajectories, sweep surfaces can be used.

The TBV eliminates the need to perform scene management and visibility determination every frame. TBV's can be used in several run time occlusion culling algorithms, as they simply replace the object geometry. However, the method's efficiency is dependent on the underlying culling technique, and insertion overhead. Unfortunately, current dynamic entity visibility culling algorithms don't offer lightweight precomputation based solutions for urban environments with unrestricted viewpoints. In addition, the TBV validity period depends on viewpoint motion: it can be determined accurately only for a static viewpoint.

4 MANAGEMENT OF DYNAMIC REAL WORLD ENTITIES

A real city is populated by pedestrians, bicyclists, cars, public transportation and possibly other types of vehicles. We consider all these entities as *detail objects*, which do not contribute to visibility. In the following, we develop a view independent dynamic entity management and culling scheme using predetermined visibilities.

4.1 Topological entity management

We assume that all real entities in an urban environment are restricted to areas or paths, which can be extracted from existing map databases. As a consequence, we also assume that only physical positions are updated, not virtual, freely flying cameras. Pedestrians use sidewalks and walk in parks or marketplaces, while vehicles are restricted to essentially one-dimensional streets and occasional parking lots. Taxis, buses and bicycles can have their own dedicated lanes, and trams and subways use rails. Furthermore, public transportation is generally limited to predefined routes.

In the context of increasing GPS accuracy, [Cuy03] proposes to constrain vehicle paths to streets. Following this idea, we create topological street networks (possibly separate for each entity type, if suitable data exists), connecting areas such as parks and marketplaces to the network. We limit potential navigable areas to this network of street segments (figure 2) and areas (figure 4). Similar work has been done in [Whi07], with the addition of indoor topologies, for navigation purposes.

The resulting network consists of nodes (crossings and areas), edges between nodes (street segments), and can be viewed as an *adjacency graph*. We use *incidence lists* as internal data structures. Such a list contains pointers from nodes to adjacent edges and vice versa. Street segment geometry is stored to edges, and area geometry to nodes. The position of an entity can now be given by an edge ID, and the one-dimensional position along the street segment, or by a node ID, and the two-dimensional position within the area. Nodes that are associated to crossings do not hold area geometry.

A dynamic entity is now managed by tracking its position within this topological network. For initial placement, a quadtree provides a good external structure to localize closest street segments and areas. We

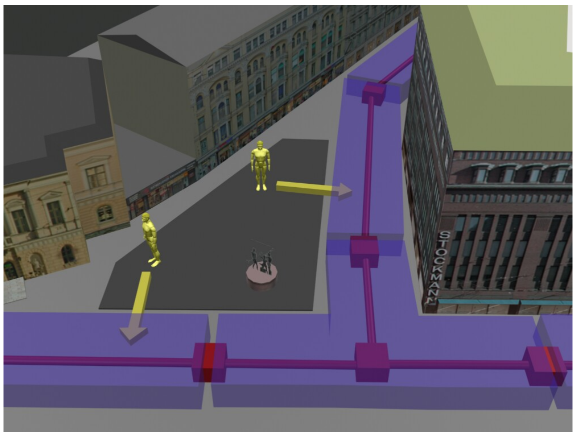

Figure 2: Pedestrians projected onto a street network.

then test if the entity lies within one of the closest areas, and if not, project it to the nearest street segment. When the entity moves, its position on the network is constantly verified. Each entity holds its locally measured position (for example, a GPS position) and the inferred position within the network.

4.2 Visibility cells

To build a system where dynamic entity visibility can be predetermined, we create static virtual visibility cells reflecting the geometry of street segments and areas. For any visible cell, we assume that entities occupying them are visible as well, in the cell-to-cell visibility manner. In pursue of conservativeness, visibility cells should cover all occupiable space. In practice, we use a constant value for cell width, estimating the widest possible street. For the height, we use the height of the tallest possible entity. We construct the cells as sweep volumes along the street geometry. For each area, a single large visibility cell is constructed as an extruded volume, again using the height of the tallest entity (figure 4).

We observe that generally crossings seem to be more visible than the streets at urban canyons (the antipenumbra of crossings tends to be larger than that of the streets between buildings). Even if a piece of a cell would be visible at a junction, the entire street would be deemed visible. Therefore, we make a minor modification to the shape of the virtual cells, and the network, where the nodes would otherwise be located only at the centers of crossings and areas. We split each street segment near a crossing, and use the short pieces to create a virtual cell reflecting the junction geometry. For example, in a T crossing, three short segments are connected to one node, constituting a single T shaped visibility cell (figure 2). An entity on any of these small segments would occupy only this particular visibility cell. We store pointers to the edges and nodes in the associated visibility cells.

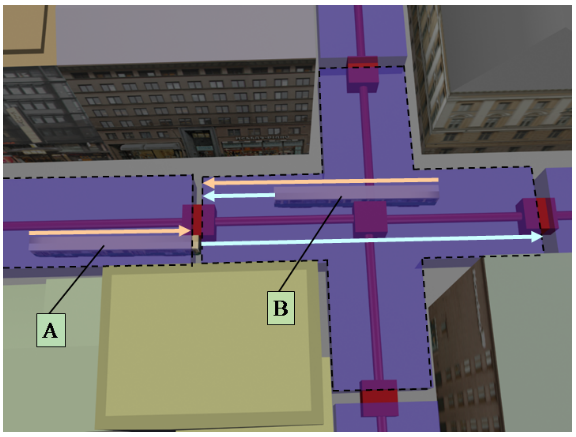

Figure 3: Cell occupations and validity periods for trams.

Results from the visibility calculations are stored into each view cell's visibility list. These lists can be downloaded in advance, or progressively streamed at run time. Visibilities are updated as the viewpoint moves from a view cell to another. For each visible cell, the occupying entities are rendered.

4.3 Approximating entity motion: cell occupation validity periods

In the sense of temporal bounding volumes, we could approximate entity motion by *discretized temporal bounding vectors* along the one-dimensional street network. A moving vehicle would be described by a larger set of points than a stationary vehicle. However, our visibility scheme utilizes preprocessed static cells. Thereby, we are primarily interested in potential cell occupation instead of the potential motion vector. We abandon the temporal bounding vector calculation, and estimate *cell occupation validity periods*. In another words, instead of estimating a new position based on a given time interval, we estimate time of travel, given a distance (shortest remaining street segment within a visibility cell). We compute separate periods for entity's front (entering a new cell) and aft (leaving the current cell). After the shorter of these validity periods expires, occupation approximations are recalculated, and visibilities reassigned.

In figure 3, the front validity period of the tram A has expired, and new periods calculated. We possess a priori knowledge of the tracks, and know that the tram will continue straight across the junction. Therefore, the front period becomes rather long, and we select the aft period. Until this period expires, the tram will occupy two cells. Tram B has been GPS positioned before its period expired, caught before reaching a node. For this tram, the front period is shorter, and selected for validity period calculation. Until it expires, the tram occupies the X shaped crossing cell.

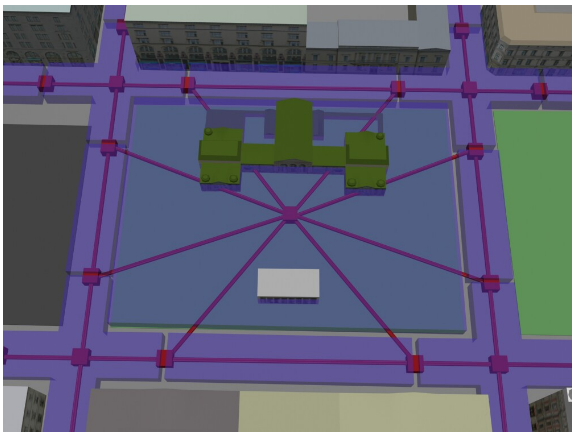

Figure 4: A single large visibility cell representing a walkable area, joined to a surrounding street topology.

4.4 Networking: subscriptions

In a peer-to-peer network, utilizing visibility efficiently is difficult. In order to know which clients are visible, a client should send position queries to every client, and do this frequently. We choose a client-server approach to allow centralized message passing with interest management. A server maintains master tables of client states, including cell occupation, validity periods and *subscriptions*.

During normal operation, clients subscribe to visible cells, and cancel subscriptions to invisible ones. This can be performed as a single subscription difference message. We also support direct subscriptions. If clients are interested in individual entities, such as other users or certain vehicles, they can subscribe to them. The server will then update these entities independently of visibility or distance. Maximum limits are set to avoid overuse. Clients are also allowed to perform *entity queries* to find, for example, their buddies, or plan their schedule based on the location of an arriving bus. Again, temporal limits suppress overuse.

Should the visibility change rapidly due to viewpoint movement (such as when the viewpoint first elevates and then descends at rooftop level), a client may receive data for already invisible entities due to network latency. This data can be stored for validity periods, and used to simulate the entities immediately if the related cells become visible again. When validity periods expire, these entities are deleted.

4.5 Computational scalability

We proceed to utilize a key notion to optimize our system to be computationally scalable. Client positioning is based on local devices, such as GPS's, so smart clients must be trusted to some extent. In addition, the real world resolves possible near parallel actions, asserting consistency, and the real entities take care of their long term situation, asserting persistency. We now assume that our clients are honest, and let them compute their position on the topological network. They also publish their dead reckoning scheme, so other clients can extrapolate their position. This is utilized in the Player/Ghost manner, so that a client sends its position and cell occupation updates only when its locally computed extrapolated position exceeds an error threshold. When a client senses unexpected, emergent behavior, such as stopping, it can immediately send a state update. Updates are also triggered when new validity periods are calculated.

In our scheme, a server can avoid dead reckoning and position projection computations altogether, simply updating its master tables, and forwarding state changes when ever it receives such data from the smart clients. For entities where our client software is not installed, such as public transportation with external tracking network interfaces, separate proxies can be used to scale the system up.

4.6 Networking scalability

When the amount of concurrent users reach millions, the local network at server side may become congested, despite our optimization efforts. We overcome this by limiting servers spatially, and increasing the number of servers, which are distributed to different subnetworks. Neighboring servers subscribe to each others' cells that lie at their shared border. If the density of users is very high, local mobile networks may become a bottleneck. In this case, motion extrapolation can be extended to cover several visibility cells, although crossings pose a problem, unless a priori knowledge on entity paths is available. For example, public transportation usually follows static routes, but pedestrians may choose any direction.

4.7 Privacy

Privacy issues are addressed by a buddy system; users can choose whether they publish their identities or not, and their target audience. However, unless they publish their identities, they will not be able to identify other users.

5 DYNAMIC ENTITIES: A REAL WORLD CASE

We have utilized our developments for a real case. We gathered map data, public transportation schedule data, and obtained access to an interface for a public transportation tracking system. For practical purposes, we expected to use GPS for positioning, with accuracy of 5m or worse, and update rates of 1Hz or less.

The system was built upon our current mobile 3D map system, the *m-LOMA* platform. m-LOMA utilizes regular 3D view cell subdivision and precalculated cell-to-object visibilities and contribution culling for static geometry, with building façades and roofs as atomic objects. Our city model consists of about 200 individually textured and 100 flat colored buildings, from the city center of Helsinki, Finland. The texture detail varies between 10–20cm. The model runs at 30–200fps in recent smart phones with 3D hardware support, such as the Nokia N93 and N95, simultaneously rendering up to 50 textured buildings, and 50–100 flat colored, distant buildings. The system relies on explicit memory management at run time, optimized for LOD textures. The 3D models and textures can be progressively downloaded over mobile networks using a pipelined binary XML protocol over TCP. The engine, its network scheme and performance are described in [Nur06, Nur07]. The presented dynamic entity management system replaces the early system described in [Nur06].

5.1 Map data and public transportation

Our map data covers street geometry, building outlines, parks, etc. Area data is given as polylines. Only the centerlines of streets is provided. No sidewalk data is available. The data is not topological and contains errors. After manual cleaning, a topological, slightly simplified street network was created to cover the city center. Visibility cells were instanced based on the resulting network. On average, 5–6 cells were needed for each city block. This increased the size of our visibility lists, but not prohibitively. For a geometrically complex 3D city model, this increase would be even less significant.

Public transportation data was provided in a collection of files in a proprietary format. We integrated multiple road polylines, simplified route geometry, and extracted bus stop positions. Unfortunately, the route data was very inaccurate, randomly misaligned to the street data, containing various loops, zig-zag shapes etc. It is stored in the local public transportation organization's internal database, and exported every time a schedule change occurs, so manual editing, without access to the database, would only prove a temporary solution. The public transportation tracking system provides estimates of arrival for the bus and tram stops with a granularity of one minute. It has a SOAP interface, and suffers from a latency of several seconds. Less than half of buses, but all trams, are equipped with this tracking system.

5.2 Simulation: real entity behaviors

Entities were assigned behaviors and related parameters for dead reckoning and validity period calculations set. The basic behaviors reflect the method of movement. Our current system supports pedestrians, one-part vehicles such as buses and cars, and two-part vehicles such as trams. The related parameters define their length (including 0 for pedestrians and bicyclists), maximum speed, lateral position offsets, collision avoidance schemes, collision distance threshold, and timeouts for solving dead locked situations, including temporary acceptance of collisions and entity deletion. We also specify spawn times based on schedule data. The lateral position offset allows us to randomize pedestrian locations, and vehicles can be shifted to virtual lanes.

5.3 Simulator implementation

We implemented a simulator based on existing schedule and route data to the m-LOMA system. Public transportation vehicles were modeled, and behaviors programmed. A fast bounding box test provides collision avoidance: vehicles wait at crossings and let the first arrivers pass first. The speed of the vehicles is approximated by the distance to the next stop, given the estimated times of arrival, and limited by the maximum speed. The lateral offset for vehicles was set to 3 meters to emulate street lanes.

The proprietary schedule and route data were stored to compact files, providing ETA's for each stop at each route. At run time, a Python script, installed at a proxy server, queries the SOAP positioning interface a few times a minute, discards the overhead (over 95% of the data), and forwards only those ETAs for vehicles that are actually traced. A set of timeouts are frequently used to resolve collisions. Vehicle motion is not always parallel to actual streets due to the route data inaccuracies. For visibility cell occupation determination, vehicles on these routes are projected to the street segments, which were parsed from the more accurate road database. These two data sets do not coincide everywhere, so currently we run the simulation with buses and trams placed on the route data set.

The first version of the system has been running in a local science park on a desktop computer (see figure 1) consecutively over a year. We have recently ported the system to mobile devices. At most a few dozen public transportation vehicles occupy our modeled city center at a time, which is no burden to our system, especially due to the poor granularity of tracking. Our system allows tracking and distribution of position updates of the entire local public transportation fleet, but visibility culling is meaningless outside the 3D modeled area.

Rendering the scene at full speed in a smart phone consumes batteries fast. We have implemented a configurable tick rate to scale power consumption down. In addition, a separate toggle button can be used to pause the entire simulation.

6 CONCLUSIONS AND FUTURE

We have presented a lightweight and efficient visibility culling and message reduction mechanism for dynamic entities based on precalculated, static visibility cells, exploiting geometrical properties of urban environments. Relying on the real world to solve consistency issues, the scheme combines the best parts of P2P networks and client-server architectures: positioning computations and decision making processes are performed at client side, while a server manages global visibility and subscription look-up tables and acts as a fast state update passing switchboard. The system is truly scalable, as the only potential bottleneck, server-side networking, can be extended by spatially limited servers, which exchange borderline data. However, situations where a very high number of dynamic entities occupy a small area pose a problem. If a viewpoint is high above ground level, looking down, visibility optimizations are of no use, and both mobile networking and local rendering resources become a bottleneck.

Precomputed visibilities exchange run time computations to larger memory consumption, but the increase of visibility lists is acceptable. Even where small buildings cannot occlude moving entities, the drawback in not significant: the number of virtual visibility cells is still much less than the number of building façades we use for static visibility calculations.

We have implemented a mobile 3D map with progressive content download, applying an efficient XML based binary protocol, static visibility preprocessing and buddy tracking, running a near real time tracked public transportation simulation. The current public transportation tracking system does not push our system to its limits. In addition, we have tracked only a few pedestrians at a time. Our local public transportation organization is implementing a direct GPS tracking based system, which we will integrate as soon as it becomes available. Until that happens, we will perform artificial benchmarks to acquire quantitative performance statistics.

In near future, we will utilize the presented system in managing a different type of dynamic entities, namely a swarm of cleaning robots in an indoor environment.

ACKNOWLEDGEMENTS

The author wishes to thank Nikolaj Tatti for his heroic programming efforts, Ville Helin for implementing the first system, Heikki Vuolteenaho for Symbian programming, Ilpo Ruotsalainen for network programming and Mikko Rasa for initial simulator programming. This work was supported in part by EU Interreg IIIA, EU FP6 ICT, and City of Helsinki.

REFERENCES

[Air90] Airey, J. M. Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations. PhD thesis, UNC Chapel Hill, 1990.

[And00] Andújar, C., Navazo, I., and Brunet, P. Integrating occlusion culling and levels of detail through hardly-visible sets. Computer Graphics Forum, Vol. 19, No. 3, pp. 499–506, 2000.

[Ar00] Ar, S., Chazelle, B., and Tal, A. Self-customized BSP trees for collision detection. Computational Geometry, Vol. 15, No. 1-3, pp. 91–102, 2000.

[Bau01] Baus, J., Kray, C., and Kruger, A. Visualization of route descriptions in a resource-adaptive navigation aid. Cognitive Processing, Vol. 2, No. 2-3, pp. 323–345, 2001.

[Bla92] Blau, B., Hughes, C. E., Moshell, M. J., and Lisle, C. Networked virtual environments. In SI3D '92 conf.proc., pp. 157–160, ACM Press, 1992.

[Bur05] Burigat, S., and Chittaro, L. Location-aware visualization of VRML models in GPS-based mobile guides. In Web3D '05 conf.proc., pp. 57–64, ACM Press, 2005.

[Cap97] Capps, M. V., and Teller, S. J. Communication Visibility in Shared Virtual Worlds. In WET-ICE '97 conf.proc., pp. 187–192, IEEE Computer Society, 1997.

[Che00] Cheverst, K., Davies, N., Mitchell, K., Friday, A., and Efstratiou, C. Developing a context-aware electronic tourist guide: some issues and experiences. In CHI '00 conf.proc., pp. 17–24, ACM Press, 2000.

[Chh05] Chhugani, J., Purnomo, B., Krishnan, S., Cohen, J., Venkatasubramanian, S., Johnson, D. S., and Kumar, S. vLOD: High-Fidelity Walkthrough of Large Virtual Environments. IEEE Trans.on VCG, Vol. 11, No. 1, pp. 35–47, 2005.

[Chr92] Chrysanthou, Y., and Slater, M. Computing Dynamic Changes to BSP Trees. Computer Graphics Forum, Vol. 11, pp. 321–332, 1992.

[Coh03] Cohen-Or, D., Chrysanthou, Y. L., Silva, C. T., and Durand, D. A Survey of Visibility for Walkthrough Applications. IEEE Trans.on VCG, Vol. 09, No. 3, pp. 412–431, 2003.

[Cuy03] Cuyi, Y., and Ge, S. S. Autonomous vehicle positioning with GPS in urban canyon environments. IEEE Trans. on RA, Vol. 19, No. 1, pp. 15–25, February 2003.

[Far04] Färber, J. Traffic Modelling for Fast Action Network Games. Multimedia Tools Appl., Vol. 23, No. 1, pp. 31–46, 2004.

[Fuc80] Fuchs, H., Kedem, Z., and Naylor, B. On visible surface generation by a priori tree structures. In SIGGRAPH '80 conf.proc., Vol. 14, No. 3, pp. 124–133, 1980.

[Fuc83] Fuchs, H., Abram, G. D., and Grant, E. D. Near real-time shaded display of rigid objects. In SIGGRAPH '83 conf.proc., pp. 65–72, ACM Press, New York, NY, USA, 1983.

[Gre93] Greene, N., Kass, M., and Miller, G. Hierarchical Z-buffer visibility. In SIGGRAPH '93 conf.proc., pp. 231–238, ACM Press, New York, NY, USA, 1993.

[Joh87] Johnston, R. The SIMNET visual system. In ITEC'87 conf.proc., pp. 264–273, ITEC, 1987.

[Mar06] Marsh, J., Glencross, M., Pettifer, S., and Hubbold, R. A Network Architecture Supporting Consistent Rich Behavior in Collaborative Interactive Applications. IEEE Trans. on VCG, Vol. 12, No. 3, pp. 405–416, 2006.

[Nur06] Nurminen, A. m-LOMA - a mobile 3D city map. In Web3D '06 conf.proc., pp. 7–18, ACM Press, 2006.

[Nur07] Nurminen, A. Mobile, hardware-accelerated urban 3D maps in 3G networks. In Web3D '07 conf.proc., pp. 7–16, ACM Press, 2007.

[Pos02] Pospichil, G., Umlauft, M., and Michlmayr, E. Designing LoL@, a Mobile Tourist Guide for UMTS. In Proceedings of Mobile HCI 2002, pp. 140–154, Mobile HCI, Springer-Verlag, 2002.

[Prz05] Przybilski, M., Campadello, S., and Saridakis, T. Mobile, on Demand Access of Service-Annotated 3D Maps. In IASTED SE'05 conf.proc., pp. 448–452, IASTED, 2005.

[Rak01] Rakkolainen, I., Timmerheid, J., and Vainio, T. A 3D City Info for mobile users. Computers and Graphics, Vol. 25, No. 4, pp. 619–625, 2001.

[RMa95] R.Macedonia, M. A Network Software Architecture for Large Scale Virtual Environments. PhD thesis, Naval Postgraduate School, Monterey, California, June 1995.

[Sud97] Sudarsky, O., and Gotsman, C. Output-sensitive rendering and communication in dynamic virtual environments. In VRST '97 conf.proc., pp. 217–223, ACM Press, 1997.

[Tel92] Teller, S. J. Visibility Computations in Densely Occluded Polyhedral Environments. PhD thesis, Univ. of California at Berkeley, 1992.

[Tom06] TomTom. TomTom, MOBILE navigation. http://www.tomtom.com, 2006.

[Whi07] Whiting, E., Battat, J., and Teller, S. Topology of Urban Environments. In CAAD Futures'07 conf.proc., 2007.

[Zha98] Zhang, H. Effective occlusion culling for the interactive display of arbitrary models. PhD thesis, Univ. of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1998.