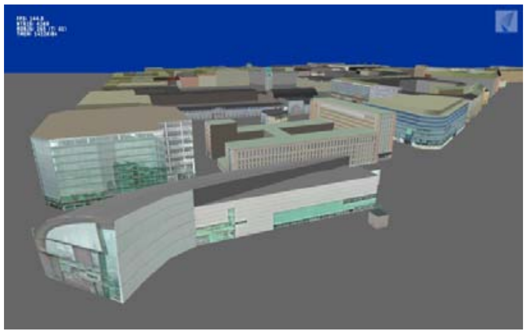
m-LOMA - a Mobile 3D City Map

Antti Nurminen[*]
Helsinki University of Technology



Abstract

m-LOMA, mobile LOcation-Aware Messaging Application, is designed to be a mobile portal to location-based information in cities. The user can perform textual searches to location-based content, navigate using 2D maps assisted by a GPS, and leave messages to the environment, or recognize the environment from a 3D map. The 3D map view is the key feature of the m-LOMA system. The m-LOMA client is capable of rendering photorealistic 3D city models with augmented location-based information in a smart phone without hardware rendering support at interactive frame rates. This paper presents the key challenges and solutions in creating this 3D map engine and a lightweight but photorealistic 3D city model.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical User Interfaces H.4.3 [Information Systems Applications]: Communications Applications—Information Browsers

Keywords: mobile computing, 3D maps, mobile guides, 3D graphics, visibility, GIS, VRML

1 Introduction and related work

Map is a representation of the environment, and used as an aid for creating a mental image of the spatial relationships of the local surroundings. It can present topological identifiers, such as street names, that can further aid high-level cognitive tasks such as wayfinding. Use of maps and cartographic conventions are strongly tied to cultures, and require *a priori* knowledge of the representations used [Barkowsky and Freksa 1997]. For example, western 2D maps are not necessarily understood in non-Western cultures. The key hypothesis of a realistic 3D map is the short cognitive distance between the observed world and the displayed virtual world, which should make it a more intuitive interface for all users, at least for simple lower-level tasks such as recognition.

Mobile guides have generally been of great interest to HCI researchers. The potential of mobile guides is acknowledged in various studies where the temporal and spatial context provide the user information that is useful now, here [Pospichil et al. 2002; Iacucci et al. 2004; Kray et al. 2003; Cheverst et al. 2000; Abowd et al. 1996]. An extensive list of recent work is provided in [Baus et al. 2005]. Certain guides are designed for tourism, where the guide can provide information on points of interest or actually guide the user through the sites [Bornträger et al. 2003; Cheverst et al. 2000]. Some mobile guides are created for messaging, or augmenting visited routes with personal information for "re-travelling" [Iacucci et al. 2004]. Certain guides are used only for navigation, but the graphical user interfaces have been adapted for different cases [Krüger et al. 2004]. Research on 3D guide capabilities has been carried out, with results indicating that the 3D view helps in recognizing local landmarks [Rakkolainen et al. 2001]. Many of the observations related to perceptual phenomena are in accordance with earlier studies of salient features and urban environments in general [Lynch 1960; Vinson 1999].

While current technology allows reasonably straightforward development of 2D map based mobile guides with textual search capabilities, there is no standardized way of creating a mobile 3D map. Before mobile 3D graphics has been possible, researchers have used alternative means to conduct studies on 3D maps. In [Rakkolainen et al. 2001], a realistic 3D city model was built, but due to slow rendering (1 frame per 8 seconds) in a *personal digital assistant* (PDA), field studies were performed using pre-rendered images embedded on web pages. In a succeeding study, field studies were performed using a more powerful laptop computer [Vainio et al. 2002]. In [Brachtl et al. 2001], pre-rendered video clips were used. On certain applications, figurative symbols or 3D graphics are rendered on top of the 2D map to ease the recognition of landmarks [Krüger et al. 2004]. While these studies have yielded interesting results on navigation and use of 3D maps, they are still simulations of a real mobile 3D map and lack many of the possible functionalities.

Very few studies exists with real mobile 3D maps on mobile devices with reasonably realistic buildings [Kray et al. 2003; Oulasvirta et al. 2005]. In the first case, the viewing distance was severely limited and a generic capability of information delivery was missing. The second study was carried on using the m-LOMA system.

[*]e-mail: andy@iki.fi

To overcome the lack of 3D map engines, studies have been made to create guides based on direct model viewing that would allow the use of a standardized model format, such as VRML, using VRML viewing software or libraries such as PocketCortona [ParallelGraphics 2005]. For example, [Burigat and Chittaro 2005] use VRML models and address the issue of information retrieval from 3D models in a promising manner.

Currently, a similar direct rendering approach is possible with the Java JSR-184 API and the M3G model format. The M3G format is a scene graph that can be directly parsed and rendered by the JSR-184 API. Early attempts to utilize this higher level API demonstrate the innate problems in rendering massive scenes: in a demonstration of [Bleschmied et al. 2005], a medium-sized city model took 2-3 minutes to load on a Nokia 6630 smart phone, and only one or two selected buildings could be textured, yet the rendering speed was below interactive speeds (1 frame per second or less). Software crashes and memory overflows were common.

2 Mobile devices and computer graphics

Before the summer of 2003 it was possible to create 3D content for mobile devices only using custom 3D development environments such as *Fathammer X-Forge* [Fathammer 2005] or model viewing libraries such as *PocketCortona* that were based on proprietary rasterizers[1]. The first standard real-time 3D graphics application programming interface (API) for mobile devices, OpenGL ES [KhronosGroup 2005], was published in July 2003. Being a subset of OpenGL, OpenGL ES is a low-level rasterization interface accessible from native C/C++ code. A Java-based API, JSR-184 (M3G) [JCP 2005], was introduced soon after OpenGL ES. This API provides higher-level functionalities and an efficient binary scene graph file format also called M3G. A M3G file can be parsed in and rendered with little programming effort. JSR-184 is designed to use OpenGL ES for rasterization. The rendering pipeline features of OpenGL ES are mostly exposed to a native C/C++ programmer, but for a JSR-184 programmer using the J2ME environment they are hidden.

3D applications are extremely resource-demanding programs. Unfortunately, the field of mobile applications is one of severe limitations. Computational resources are sparse; CPU power, memory size and memory speed are limited, mobile networks are slow, storage is usually slow or insufficient, and the displays are small. Many studies acknowledge the limited 3D computational powers of non-hardware accelerated mobile devices, and thereby consider them unfeasible platforms for rendering realistic 3D images from massive models at interactive rates [Bessa et al. 2004; Rakkolainen et al. 2001; Burigat and Chittaro 2005]. These studies also estimate that the situation will change as hardware advances.

When 3D hardware becomes generally available for mobile devices, a boost in rendering times is to be expected. But even if available computational resources do increase, so do the user expectations on the quality and complexity of the models. Therefore, relying on future hardware improvements is not likely to remove the problem. For example, the current highly optimized 3D game engines still suffer from lack of computational resources, even though the 3D graphics power available for consumers has exploded.

But even though resources in mobile devices seem to be sparse, we can create a simple evaluation method to determine if a given platform is suitable for 3D rendering, given that a 3D graphics API exists for that platform. A rendering system is called *output sensitive* if the rendering rate is directly proportional to the number of visible primitives. Such a scene can be easily constructed of a few dozen textured polygons that fit the screen without overlapping each other. The rendering rate can be measured. We postulate that *if a system can manage the rendering of an output sensitive set of primitives at interactive rates, it is ready for interactive 3D applications*. If a system can survive a simple test such as this, it is then merely a question of optimization (and storage capacity) whether large, realistic models can be run in a small device or not.

[1]Fathammer X-Forge currently uses OpenGL ES for rasterization

Based on initial benchmark results, we assume our devices will be ready for 3D applications by the definition above. We also assume a decent amount of RAM, say, 10MB, and either a wireless network connection or a local storage, say, 32MB or more. This would include most currently available PDA devices and several relatively recent smart phones, such as the Nokia n-Gage or 6630.

3 Objectives

While the early guide systems using 3D - one way or another - have provided excellent information on navigation and location-based services, they have been limited in functionalities. The technological challenge of the m-LOMA project was to create a mobile 3D application with a rich set of information pulling and pushing features, photorealistic 3D rendering quality with a large view distance, yet achieve interactive frame rates.

Several user scenarios can be created where the user is allowed only a limited viewing mode. For example, one can suppose that the environment is easiest to recognize from the first person view. In this case, the viewpoint can be forced to move only at the ground level. On another scenario, one can claim that the user can have a better overall understanding of the scene from above. Here the viewpoint can be set at a certain height, looking down in a fixed angle. Both these scenarios make visibility calculations relatively straightforward. In the first case, street topology can be used to render buildings along the current streets. In the second case, rendered buildings can be limited to a rectangular area. In m-LOMA, we allow the user to freely roam the entire 3D space. Perhaps later user studies suggest useful restrictions on viewing direction and movement, if any. We also require that the viewer should not be allowed to go inside the model; a collision avoidance scheme should keep the viewpoint outside the buildings.

The m-LOMA mobile client should be able to answers to questions ”What is ...?” and ”Where is...?”. A user should be able to point at any feature on the map and receive an explanation or a table of contents of that feature. A user should also be able to make a location-based query and receive a list of potential answers, and be routed to the selected one. The system is also required to allow the users create their own content, messages, that could be attached to any point of an object or a point in the map space, which can be viewed by friends or everybody. In addition of these *pull* functionalities, it should also be able to *push* information, such as alerts and notions on disturbances on public transportation. Dynamic information should be updated in real-time, for example GPS tracked other users and possibly public transportation vehicles.

The testbed area should be relatively large and detailed, to cover an entire city center. Real-world databases such as restaurant and tourist info should be made available and associated to the scene. GPS positioning should be supported. Network latencies and speed related to wireless data transmission should be minimized, whether it is the model, feature information, user information or positions of tracked objects that are transmitted. Long download times, from network or local storage, should not pause the application. The system should be scalable, using smart information filtering techniques such as visibility calculations at server side to limit the transmitted data. The user interface should be easy to use and allow fast operation as the mobile user has a relatively short attention span [Oulasvirta 2005].

The development aims for a multiplatform software that could run on Linux, Windows, MacOS X, WinCE (MobileWindows) and Symbian.

4 3D Engine for urban scenery

Any efficient 3D engine exploits the features of the scenery the engine is intended for. A mobile 3D engine can apply the same classical optimization techniques as any 3D application, even though from the viewpoint of very limited resources. In the following, a 3D engine for urban scenery is developed, selecting and implementing suitable *visibility culling* algorithms.

4.1 Preprocess visibility culling

Work on the architectural walkthroughs in 90's by Airey [Airey 1990] and Teller [Teller 1992] started a series of solutions to the problem of rendering massive models. Considering mainly indoor scenes of polyhedral structures, an observation was made: only a subset of the model, the *potentially visible set* (PVS), is required to render at a time. These indoor scenes can also be transformed with the *binary space partitioning* algorithm [Fuchs et al. 1980] into efficient search structures, *BSP trees*, also representing the geometry. For PVS calculations, the model space was subdivided to *cells*, such as rooms, that are separated from each other by *portals*. Smaller objects within the rooms, *detail objects*, do not contribute to the visibility calculations. This scheme assumes convexity for the cells, implying that construction of suitable models would require *constructive solid geometry* (CSG) methods, or severe restrictions set for boundary representation models (*B-reps*) [Thibault and Naylor 1987; Sudarsky and Gotsman 1997]. While being popular, the BSP/portal approach is not optimal for an outdoor scene.

For the m-LOMA system, we use a PVS approach, supporting common B-rep models such as VRML97. We divide our space to cubical volumes, cells, in all three dimensions, and perform *from-region* [Nirenstein et al. 2002] visibility culling as a preprocess. We use *meshes* as atomic objects to which visibility is performed, defined later in section 5.2. At street level, convex polyhedrae could be used as cells, fitted along streets, between the buildings [Marvie and Bouatouch 2004]. We allow the user to roam freely in three dimensions, so instead of having a specific, possibly manually adjusted solution at street level, we use plain 3D grid subdivision throughout our space. The cell size is specified to be less than the average building height. Our preprocess samples the model from a set of points inside the cell, namely from the corners and the centre, to each direction. Our solution is *approximate* [Nirenstein et al. 2002], as sampling always leaves room for error. To increase the *aggressivity* of the algorithm, we also define a *hardly visible set* [Andújar et al. 2000] consisting of objects that have less than a predefined amount of pixels visible, and leave everything belonging to the HVS out.

Using cubical cells implies one small problem for the PVS calculation at street level: some cell corners may lie inside the buildings, and if *backface culling* (see section 4.5) is enabled, the buildings nearby will be included in the PVS. Therefore, we disable backface culling during the PVS calculation. A few unnecessary backfaces will end up in the PVS lists, but are culled at runtime with negligible CPU cost. From the sky, no backfaces will be visible if the model has been correctly built.

4.2 Runtime visibility culling

Most preprocess visibility culling methods calculate the visibilities independently of the possible runtime viewing direction. In addition to visibility culling, *view frustum culling* (VFC) methods further omit polygons not directly in the view of the camera. For example, [Moller and Haines 1999] describes a selection of VFC algorithms.

m-LOMA uses a two-level hierarchy for frustum culling, applying temporal coherence. At the higher level, we use *mesh groups*, containing lists to individual meshes. During initialization, all mesh groups belonging to the initial PVS cell are loaded, and marked invisible. Then, for each group, its bounding sphere is culled against the view frustum. For all visible groups, meshes are loaded and per-mesh frustum culling performed. Visible meshes are marked. In addition, a counter is assigned to each tested mesh and mesh group. The counter indicates the amount of rendered frames to wait before repeating the test and depends on frame rate. For visible groups, at maximum 20 frames can be rendered before a test occurs again. The idea of the counter is in the low probability of major visibility changes during successive frames when moving. During rotation, abrupt changes are likely, and frustum culling is performed to all groups, independently of the current counter value. This is recursed to the individual meshes. Loading models and textures is limited per a rendered frame to maintain responsiveness. When the system is initializing and loading meshes, the user can perceive a partial model during the first frames. In principle, the initialization could take place before the first rendered frame, to hide the loading from the user.

In addition to detecting visible meshes, a mechanism is needed to solve screen-space per-pixel visibilities. The standard way is to use the depth buffer, the *Z buffer*, that is part of all modern rendering pipelines. The *Z test* lets a pixel to be drawn to the frame buffer only if it's the Z value is smaller than the previously calculated value. We can further utilize this to avoid unnecessary texture accesses that would follow the Z test in a rendering pipeline. We arrange all visible meshes in depth order, closest-first, to let the closest meshes fill in the Z values. Parts of meshes that are further away and behind closer meshes are Z tested, but not textured. The major advantage of mesh sorting is in *collision detection*, described in section 4.6.

4.3 Culling dynamic objects

The m-LOMA system supports *dynamic objects*, for example GPS tracked users or simulated or tracked public transportation vehicles. [Sudarsky and Gotsman 1997] present optimization techniques for a dynamic scene, where a *temporal bounding volume* (TBV) can be used to estimate a visibility of a moving object, and reduce unnecessary network traffic between a server and client. In m-LOMA, we use *virtual voxels*, similar in size and shape to our PVS cells and with similar grid distribution, but that lie only at street level. The voxels are included in the PVS calculation, but only a predefined amount of voxel visibilities are created for each PVS cell.

For any moving object, the current voxel is determined simply by approximating the object by a set of points, and resolving in which voxels these points reside based on coordinates. If any of the voxels lie within our PVS and are within our view frustum, we consider the object visible, and render it. For a GPS tracked user, there is only one coordinate pair (assuming the user moves slowly), but for larger, faster objects, such as a trams and buses, a different scheme is applied. Instead of a temporal bounding volume, we approximate a fast-moving object by *temporal sample points*. If any of these points is visible (within a visible voxel), the object is rendered.

4.4 Levels of detail and texture management

Objects far away occupy only a small amount of screen space, but only the minimal information of the object should be needed to convey the meaning of what is being viewed [Clark 1976]. If multiresolution versions of an object exist, a rendering system can choose the suitable version based on the estimated size on the screen, or viewing angle. These common techniques are called *level of detail* (LOD) techniques. Sometimes, an object can be represented by an *imposter*, that replaces the actual model. A common choice for an imposter is a *billboard*, typically a low-polygon rectangular textured plane that always faces the viewer. An extensive presentation of LOD techniques is given in [Luebke et al. 2005].

In addition to geometrical models, LODs can be applied to textures. A common way to enhance texture quality is through *trilinear interpolation*, also called *mipmapping* [Williams 1983]. A mipmapped texture is stored in the memory with a set of downscaled versions. When rendering, pixels are sampled from the mipmapped textures based on screen size and distance, and linearly interpolated. In our case, the limited memory of our target platforms sets strict limitations on texture usage. For example, in a common view, 100-200 meshes could be visible (see section 7). With a decent resolution, say, from 128x128 to 256x256 pixels, we would need 6-24MB of runtime memory for mipmapped textures. In practice, current smart phones can hardly allow 1-2MB.

For our system, we choose a texture-based LOD technique: we open and use *only* that particular LOD version of a texture that is comparable in size to the object size in screen coordinates. We will not be able to perform trilinear interpolation, and will suffer a bit from sampling errors when viewing buildings along the streets. On the other hand, it is highly unlikely to be very close to several buildings at the same time, requiring a large amount of textures. Using this technique, we can allow the use of high-resolution textures in smart phones. In addition, we will define a limit, the lowest LOD level, after which texturing will be disabled and a plain color used.

4.5 Rendering optimizations

The 3D rasterization pipeline properties can be addressed from a 3D API that exposes them to the programmer. Appropriate measures can be taken to increase rendering speed, or more commonly, to avoid slowing it down. The 3D pipeline constitutes of a long series of calculations from 3D transformations and lighting to rasterization, per-pixel Z evaluation and texturing (or shading). This pipeline acts as a state machine, and needs to be restarted when a state change is issued. As a generic rule state changes are to be avoided. The pipeline should also be kept as short as possible by turning off all unnecessary features. For example, in m-LOMA, we do not use vertex lighting. Our mesh sorting scheme also avoids unnecessary texture accesses (section 4.2).

The pipeline can also perform *backface culling*, avoiding rasterization of polygon sides that are facing away from the viewpoint, given that the normals of rendered polygons are arranged in a consistent manner; all the building wall normals should point outward, or inward, but not randomly to both. Due to the minor issue on street level visibility preprocessing (section 4.1), a few backfacing meshes are included in PVS lists. We enable backface culling to discard these meshes at run time.

All adjacent geometry fed to a graphics pipeline should be organized into connected triangles, namely triangle strips or triangle fans, to avoid unnecessary transformation of individual vertices. Certain pipelines may keep a small vertex cache to speed up the rendering of individual but adjacent triangles, but a better strategy is to organize the geometry in advance. This problem typically arises with storage formats that represent geometry in a form that are easy to interpret into individual triangles, such as VRML's `IndexedFaceSets`. In cases where geometry data sets are large or run time resources sparse, it is advisable to arrange model formats to close to binary-compliant geometry representations to avoid unnecessary parsing at runtime. The m-LOMA preprocessing stage attempts to create as long triangle strips as possible from the `IndexedFaceSets`, and packs them in a binary format.

[Miettinen 2005] provides a good general-purpose view on real-time rendering optimization, given in the context of mobile 3D graphics.

4.6 Collision avoidance

Collision avoidance techniques are used to prohibit the user from going through walls, which has been found disorienting [Smith and Marsh 2004]. The calculation involves intersection tests between the line along which the user is moving and the polygons of the world. If a collision is detected, the intersection point is calculated, and the viewpoint is constricted to be outside of the colliding polygon at a predefined minimum distance. A straightforward calculation does not scale up well unless the polygons are efficiently organized. In our case, we depth-sort the meshes during loading. We test the bounding sphere of the meshes in our sorted visibility list against the bounding sphere of our movement vector. For meshes that succeed the test, more careful intersection tests are calculated for the possible colliding polygons. The amount of potentially colliding meshes is typically small. For example, if the user is on ground level and near a corner, this list could contain 4 meshes, requiring raycasting towards maybe 20 triangles. Commonly, when moving towards a normal building, only one mesh (the wall) is colliding, consisting of two or four triangles. The worst-case scenario would be in proximity of a corner of a high-polygon model.

At this point, we have not yet decided about using topographical data. We prepare a simple lookup scheme for finding the current ground triangle, if any, and prepare an interpolator for retrieving the current elevation from the vertices. The collision detection against the ground is calculated independently of other collision avoidance schemes.

5 Modeling cities

Let us loosely observe a common city view. The environment is dominated by large structures, occluding each other especially when viewed from the ground level. These structures, buildings, are unique. They have been designed by a legion of architects since the founding of the city and exhibit individual characteristics, often typical to the construction era (in Europe, several hundred years old buildings may me adjacent to modern ones). The overall, average shape of the buildings appears to be rectangular. There is geometric detail, but it is typically small in scale in comparison to the size of the buildings. After these hopeful notions we immediately recognize also medium-sized geometrical features, for example structures and shapes near the roofs.

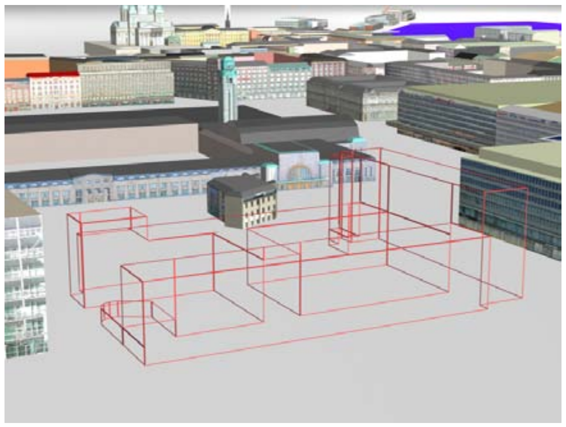


Figure 1: Modeling for mobile devices using lightweight geometry.

Our test scenario, the city of Helsinki in Finland, consists of buildings of relatively the same size, only 3-6 stories tall. There are no skyscrapers, for example. The few tall buildings that exist are clear landmarks. But there are other landmarks as well: for example, statues and various other artwork that are commonly present in urban environments. In addition, building façades are often populated by various ads and company logos, adding clearly recognizable detail. Even though the salience of such features is a subjective matter, our goal is to carefully reproduce as many *potentially salient* features as possible. The roads exhibit less direct salience: the asfalt, sidewalk, curb stones and traffic signs tend to be similar everywhere. On the other hand, these are topological cues to the environment. The traditional street maps almost completely rely on the topology, the network of connected and individually named streets. Very few street names can be directly guessed from the look of the street; they are marked separately in the environment.

5.1 m-LOMA modeling methodology

We consider our observations on our case area, and prior research on urban areas and navigation [Lynch 1960; Paay and Kjeldskov 2004; Vinson 1999] in our goal to recreate a recognizable urban environment. We need to include distinctive features from the urban environment: building geometry, individual building façades, landmarks and other potentially salient features such as statues, parks, shores and routes. Based on the presented simplification and optimization methods, we set up a modeling methodology that would efficiently reproduce these features.

There are two available 3D models on our case area: a large model created to be a web portal [Linturi et al. 2000] and a municipal-administered accurate model. The first one is a smart model, manually created to optimize recognizability with geometry, with various levels of detail. Unfortunately, this model has no textures. The second case is a municipal database, very accurate but with varying design (some models are created with CAD tools, some automatically from aerial laser scans). Many buildings have textures, but the texture quality varies.

In computer graphics it is more efficient to introduce detail with textures, not complex geometry, which requires far more computational power. The photorealism of a city model should therefore be based on textures, not geometry. Our earlier experience with city modeling, for example for [Kray et al. 2003], has shown that modeling photorealistic cities for mobile use is fast, given that only a lightweight geometry is required. On the other hand, creating textures is more laborious. As neither of the available models provide consistent good-quality textures, and the geometry is not directly suitable for mobile use, we choose to create our own textured lightweight model, possibly adapted to the requirements of our engine.

Figure 1 presents a lightweight design for a building. We use digital photography for texturing. We choose to omit geometric LODs and trust on textures for recognizable detail. As rooftops are hardly visible to the street level and contain little salient detail, they are not textured, but assigned a color.

Landmarks are recognized as critical navigational aids [Darken and Sibert 1993]. In addition, [Pasman and Jansen 2003] evaluate methods for geometrical or image-based simplification of objects in networked environments, and note that at long distances, objects can be simplified to simple imposters, billboards. To enable navigation with known and visible landmarks from long distances, we choose to create a set of billboards from selected buildings such as churches and a railway station. Such landmarks will be rendered separately, independently of our visibility range. A few landmarks far away from our 3D modeled area are created as billboards, but not modeled.

5.2 Modeling in practice

To verify that surface normals remain coherent for backface culling, we created simple building models, exported them to VRML and analyzed the results. We found out that at least 3DMAX v.7 and its VRML exporter produce surfaces in a deterministic manner. We also needed to decide the atomic primitives we will consider individual objects in our PVS calculation. The easy alternatives are individual triangles, a collection of triangles under a `Transform` node, or hierarchies of `Transform` nodes, such as buildings and building blocks. Using individual triangles would yield almost intolerable amount of visibility entries. Using blocks or buildings would be inefficient from the visibility point of view: a small corner of a wall would lead to rendering of an entire building or a building block. As an attempt at balancing visibility list sizes and simplicity we select individual walls and roofs as identifiable meshes. For frustum culling purposes, they are arranged in named groups (see section 5.1. For untextured objects such as roofs we allow larger mesh sizes. Figure 2 presents a sample building, and its decomposition of meshes.

We use existing digital city blueprints to provide decent accuracy for the model placement. The building models are constructed based on reference photography. We choose not to model streets and sidewalks manually due to the lack of individual salient features, but hope to obtain accurate street databases that could be utilized for street detail visualization. We note that in our case, certain streets are covered by asphalt, some with smooth rocks. We prepare for automatic street texturing and create texture templates for each street type. We also create texture templates for tram rails, curb stones, etc. We hope to be able to extract street names from an existing street database.

We attempt to create clean textures in a reliable manner. We take care that the window rows and columns in the model match reality. We clean the textures a bit to remove some of the possibly occluding wires, cars, trees etc. This is done in image processing software by copying pixels from visually similar areas on top of the artifacts. To minimize the amount of individual textures, we attempt to use shared textures among building walls, if possible. We acknowledge the difficulties in reproducing the street level shopping windows as their content varies quite often, and recognition would require very highly detailed textures. We note that the street level should be considered a separate layer of the model with dynamically updated textures. We choose to use static textures for entire façades, covering also the street level. Statues are created from digital photographs by separating the background by a transparent mask.

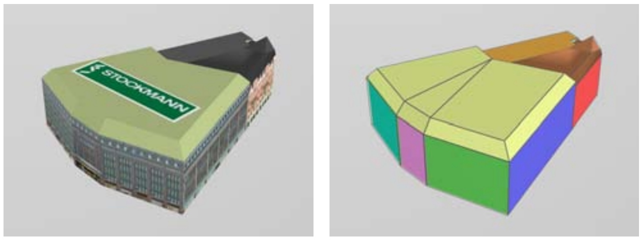


Figure 2: Splitting a building into separate meshes for visibility calculations.

We model large distinguishable land areas such as parks with flat colored geometry. In case of parks, we populate the area by billboard approximations of trees, and use green planes as the ground. Sea is similarly approximated as a set of blue planes. For local salient cues, statues seem a logical choice. [Pasman and Jansen 2003] proves that such objects should be represented with geometry, not imposters, at short distances. Unfortunately, we lack modeling resources to properly recreate them. We make a design decision to create statue models based on imposters. As an attempt to minimize the error, we use higher resolution textures on statues than on buildings.

To associate database entries to a 3D model, an anchoring mechanism is required. Our system will contain information associated to static or dynamic targets. Our static objects being statues and buildings, we group the meshes of each such object to *mesh groups*, and assign them individual names.

We choose to omit manual modeling of topography. We hope to be able to adapt our 3D models to topographical data.

We subject all our design decisions regarding selected landmarks, use of billboards, geometry/texturing ratio of the models etc. to future field studies.

To further optimize model subdivision for a PVS engine, an interesting approach would be to use automatic means to detect potentially visible areas, and split or merge the model to a view-dependent set of pieces. This would yield a larger number of individual objects (and larger total size of geometry), but the amount of entries in visibility lists would probably remain the same (a wall would normally require one list entry, but so would only the visible piece of a wall). A simple simulation of finding probable splitting axis is presented in figure 3, where a 360° degree wide and 30° high arealight has been used to illuminate a part of the scene, estimating the average roaming space of the user. The darker areas below the red line would be unlikely to be visible to a user far away.

6 Implementation

6.1 Software development

To access the 3D rendering pipeline and generally be able to implement our visibility algorithms, we used the OpenGL ES, and the

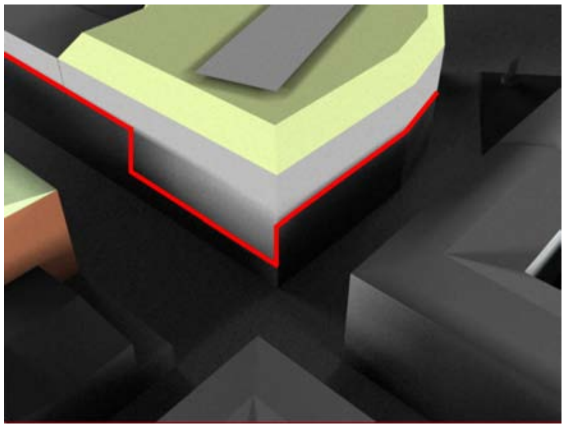


Figure 3: Possible automatic model splitting based on potential visibility.

C language. We used an implementation of OpenGL ES supporting the *Common* profile. The system was developed under Linux and ported to Windows, Mac OS X, WindowsCE (MobileWindows) and Symbian. Of these systems, Symbian was clearly an exception. Even though a unified source tree was maintained, Symbian-specific limitations sometimes dominated the development. Porting OpenGL ES to the various platforms was straightforward, while the entire application needed to be suited for Symbian's restrictions.

A lightweight OpenGL based widget set was developed for platform-independent graphical user interfaces. With Symbian we used the native Symbian Series 60 user interface system.

The development of the m-LOMA system consisted of essentially three parallel stages; modeling the environment, programming the preprocess tools, and creating the runtime environment consisting of a server, database and the client.

6.2 3D preprocessing pipeline

The m-LOMA 3D model preprocess pipeline, implementing the algorithms discussed above, has five stages:

1. VRML parsing
2. Texture processing
3. Visibility calculation
4. Visibility list encoding
5. File packaging

VRML parsing. The VRML scene is parsed for only a few features existing in our scene graph. We separate high-level `Transform` nodes that contain group names, and extract the child objects under the next `Transform` nodes, where the geometry is described as `IndexedFaceSets`. We create triangle strips out of the individual triangles in `IndexedFaceSets`. During parsing, all meshes are assigned an individual identifier. A separate file is created for associating group names to these identifiers. We also extract the object textures. We will use shared textures, so only one entry per texture is created.

Texture processing During the preprocess stage, LOD versions of textures are created and stored in separate files in a compressed format. As the LOD creation is an offline preprocess, we have time to do further analysis on the textures. For example, we observe that building façades exhibit repeating geometric patterns (see figure 4). These *templates* could be extracted, and a rule set constructed to build the original image from these templates. Single templates can be compressed with common image compression methods. With clean façades, this method may prove quite efficient. Our initial implementation results support this hypothesis.

During preliminary field studies, we observed that the common practice of using the average texture color for the lowest level LOD produces sometimes colors that do not match the perceived building colors. In a smart phone, the poor display resolution causes buildings at relatively close to fall to the color approximation, even though in reality, the building can be still be perceived accurately. A further observation is that the perceived color in reality is that of the walls, not the reflecting windows. We suggest a new approach for assigning a color for a building's lowest LOD. We analyze the texture and select the most dominant color, the color that has highest probability to appear in it. We use a straightforward solution and pick the RGB color with the highest pixel count, culled to 4 most significant bits per component. In figure 4, a sample façade is presented, with the averaged and dominant color. Our initial tests are encouraging. A quick user study with a question "which color best describes this building" consistently gave the dominant color as the better alternative, even though on many buildings the difference wasn't so noticeable.

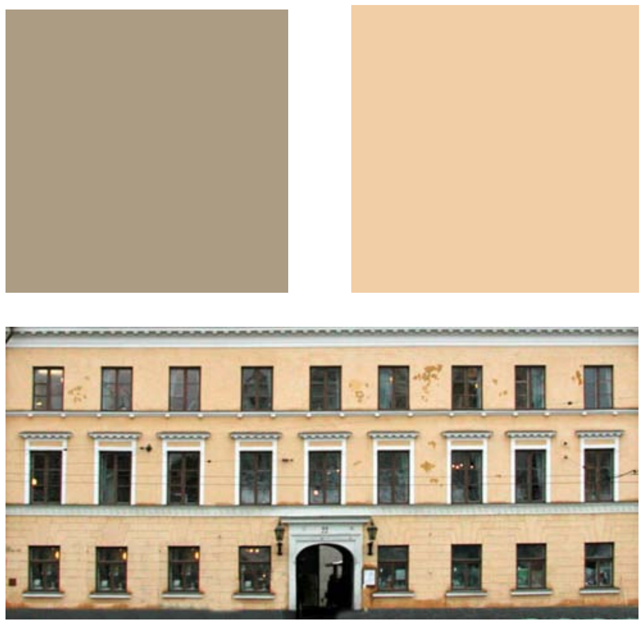


Figure 4: Selecting the color for the lowest LOD: a) (upper left) the average texture color or b) (upper right) the dominant texture color.

Visibility calculations Based on an approximate cell-to-geometry PVS algorithm, the entire space is traversed. For each cell, the scene is rendered with hardware acceleration at cell corners and the center, to every direction. This calculation is distributed over network. During rendering, unique colors are assigned for each mesh, with flat shading and no antialiasing or oversampling. We assume that only buildings contribute to the visibility. Statues and billboard-represented database entries are considered detail objects and are not rendered as occluders. To identify visible detail objects, we render them in a separate pass without Z buffer updates, using an approximate bounding box. After everything has been rendered, the resulting framebuffer is read back, and colors mapped back to id numbers. Redundant id numbers are removed.

Visibility list encoding The lists resulting from the visibility calculation stage are huge. For example, a 200x200x50 grid with an average of 500 visible objects, described by a two-byte identifier, would yield a list requiring 2GB of storage. Fortunately, especially above the roof level where the lists become large, the spatial coherence between cells is also largest. In addition, we limit the maximum height for visibility calculations to 100m. [van de Panne and Stewart 1999] describes a currently popular visibility compression scheme, based on a boolean lookup table. Even though it has been improved by, for example, [Bouville et al. 2005], it overlooks the natural three-dimensional visibility coherence. [Chhugani et al. 2005] encodes the identifier difference ΔI in three dimensions to clusters. Our approach is similar. We create visibility clusters, tree-like structures, with the following algorithm:

1. Find a location for new root node.
2. Starting from the adjacent cell with minimum difference, add a new cell. Mark a return path to the current cell into an external structure using 3 bits (dx,dy,dz).
3. Iterate (2) for the current branch until maximum difference is reached, maximum depth of branches is reached or maximum total size for a tree is reached.
4. Go up in the current branch and continue (2). When all branches for a mother node are found and maximums reached, find a new mother node (1) until all cells are added to trees.

As of this writing, further compression using Huffman encoding is being implemented, similar to [Chhugani et al. 2005].

File packaging Geometry files are stored in a compact binary format, preserving the mesh id, dominant color, vertex coordinates, vertex normals and a bounding sphere for frustum culling. For texture formats, m-LOMA supports JPG, PNG and our rule-based compression method.

In our experience, file systems used in mobile devices do not scale for situations with multiple, possibly thousands of files in single directories. Accessing a file, such as a texture, in such a directory might take several seconds. To overcome problems with the file systems, we pack our geometry, texture and PVS clusters into indexed archives.

6.3 Runtime system architecture

The overall system architecture of the runtime system is presented in figure 5. The databases are populated by python scripts that parse a set of ASCII files (a tourist database and a restaurant database), extract street names and assign preliminary coordinates. The entries are associated to the 3D model using an extension to the m-LOMA client, the *associate mode*. For each entry, the administrator assigns a correct point within the model. This association is then transmitted to the database. In addition to location-based tourist information, the database contains user information and user-specified location-based messages. Geometry, textures and visibility lists are also stored. Users are required to register to the system using a web form.

Wireless networking The m-LOMA system was designed for wireless remote rendering on low bandwidth networks, such as GPRS. The true bandwidth of GPRS varies, but typically it is as low as 5kB/s. The common latency, or round-up-time, is more than 500ms. [Schmalstieg and Gervautz 1996; Hesina and Schmalstieg 1998] describe remote rendering systems with demand-driven approach for geometry transmission, including level-of-detail models.

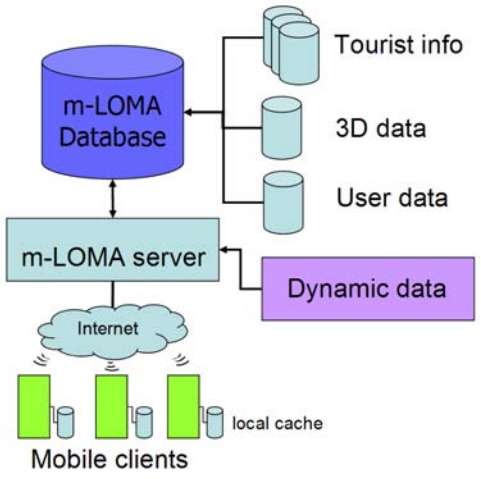


Figure 5: The m-LOMA system.

The transmission will take place for *areas of interest* (AOI's) defined by a radius. A prefetching mechanism transfers geometry in advance. [Pasman and Jansen 2003] evaluate methods for simplifying small objects with geometric or image-based algorithms for network transmission. The environment is intended for *augmented reality* (AR) applications, assuming very high network speeds (2-10Mbit) and low latencies ($<$100ms). View-dependent simplification methods for detail objects are described. In our environment, the only possible models for such simplification are the statues, but due to our lack of modeling resources, we have created them as billboards. We choose a networking approach somewhat similar to the demand driven approach in [Schmalstieg and Gervautz 1996], even though geometrical levels of detail are not in use.

Instead of an AOI, the m-LOMA utilizes the PVS scheme for determining visibility. During initialization, if a network connection is open and networking enabled, the client sends the server the current cache content, the content of the local index files. The server has full information regarding visibility and required data based on PVS precalculations. When the viewpoint moves, this position is transmitted to the server. When the server notices that new data is required, it starts sending data. First, if new PVS cells are needed, a whole PVS cluster is sent. Until this is received, the client continues rendering using the last known PVS cell contents. Possible lack of data will not affect the responsiveness of the client. Then, if geometry is needed, it is sent as individual meshes. As the third stage, if textures are missing, the server starts sending them. This data is integrated to the scene on the fly. As the PVS lookups are fast, the server could serve hundreds of simultaneous users.

When a client receives data from the server, it immediately stores it on local cache, and updates the corresponding index files. No geometry or textures are transmitted twice, unless sending fails for some reason. The whole model, including textures, PVS clusters and meshes require less than 10MB (for PDA and smart phone sets), so we assume that in normal conditions this can be saved on local storage.

Tourist content is pushed to the client based on AOI, and the client determines the visibility locally, possibly utilizing the voxel approximation. For dynamic data, data is similarly pushed to the client, and the client determines the visibility locally.

On field tests, the responsiveness of the client was verified to be adequate. Network transmission did not affect the rendering speed, and buildings appeared steadily, at slow pace. Unfortunately, due to the very slow GPRS connection, an uninitialized cache takes several minutes to fill, for example when the user rockets to the sky. As the model data is essentially static, the local caches can be easily filled in advance, by downloading the appropriate files. With a BlueTooth connection, this is rather fast. Due to the more practical and fast local file download, the support for model data transmission is currently disabled from the server.

The communication is based on TCP/IP. The routines used for accessing the network have been abstracted into platform-independent and platform-dependent components. The protocol data units are described by an XML description, from which an efficient binary protocol is generated.

6.4 3D rendering in mobile client

The m-LOMA system attempts to move the bulk of computations to the preprocess stage. As a result, the actual rendering pass is relatively straightforward:

1. Based on current position, select a PVS cluster
2. Determine the current mesh visibility list
3. Determine the currently visible messages and dynamic objects using predetermined voxel visibilities
4. Perform frustum culling to all objects
5. Assert suitable LOD textures
6. Render landmarks with Z writes disabled
7. Render meshes, billboards and dynamic objects
8. Render the user interface components

To keep the 3D map interactive, no more than one texture is loaded per a rendered frame. Texture loading is not threaded, but a list of missing textures is automatically updated. In case we run out of memory, we immediately release memory from within the software using our explicit memory management system. For desktop systems, an additional skybox presenting a sky can be rendered prior to anything else.

6.5 Memory management and caching

PVS solutions follow the tradition of CPU-memory tradeoff by releasing necessary online CPU cycles by offline computations, but increasing online memory consumption. Unfortunately, memory in mobile devices is also limited. Textures, geometry, and especially visibility lists require memory. To minimize network traffic or local storage I/O, we use data caching. Textures, PVS trees and geometry are kept in a cache. This cache memory can be pre-configured and explicitly managed by the m-LOMA client. When more memory is needed, the *least recently used* data sets are immediately released.

7 Results

7.1 Model statistics, efficiency and rendering speed

Our target platforms include laptops, PDA devices and smart phones. The laptop used in our tests was a Dell M60 with Pentium M 1.7GHz with a Quadro FX Go 700 graphics chip. The PDA's used were Dell Axim X30 and X50v, which is equipped with the

Intel 2700G 3D chip. Due to lack of support for the OpenGL ES Common profile, the X50v was tested using the 320x240 resolution and no hardware acceleration, resulting in similar performance as the X30. For smart phones, we used the Nokia 6630 and Nokia n-Gage. The Nokia n-Gage has a poor rendering performance in comparison to 6630.

The Symbian executable size is approximately 200kB, and the Windows executable about 400kB. The memory usage is 3.6-5MB in mobile devices, depending on the allowed cache sizes. In Windows, the usage is between 12-16MB.

3D Model Our final model of the city of Helsinki contains 183 textured buildings and 101 nontextured buildings split to 1029 meshes, within a 2x2km area (1.25x1.25 miles). Most of the objects, such as building façades, contain only 2-4 triangles, but certain nontextured but geometrically detailed landmark buildings are quite large (200-400 triangles). The total triangle count of the entire model is 12610. The original VRML file size is 14.3MB, and the binary geometry mesh archive size is 714kB. There are 471 individual textures, of which 426 are building textures (mostly façades) and 14 statue billboards. If all the textures were to be opened, such as in direct VRML model viewing with mipmapping, they would require 100MB texture memory. The model was created in two phases, and took roughly 8 months to create by one person. Approximately 75% of modeling time went into creating textures, 25% to the geometry. 6 presents a screenshot and a photograph of the same target. The model contains no topography. The ground is not textured due to lack of suitable map data.

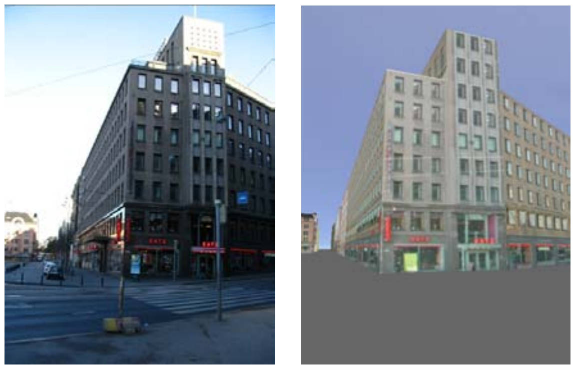


Figure 6: Accuracy of modeling. Left, real world. Right, m-LOMA.

Preprocessing For the model preprocess pipeline, we developed programs for VRML parsing, PVS calculation, PVS clustering, texture LOD creation, texture compression and dominant texture color selection. The pipeline can be configured to output versions suited to a range of devices, from high-end desktop to smart phones 1. The PVS cell size, hardly visible set pixel threshold and framebuffer resolution can be set according to the required accuracy and efficiency. The template-based texture compression is optional. Our compromise is to use 20x20x20m cell size, a little less than the average building height in our scene, and only 1 pixel HVS limit.

Table 1 provides approximate preprocess results for given platforms. The Nokia 6630 is considerably faster than Nokia n-Gage, and we can use the same data sets for both 6630 and PDA devices (the "PDA/Smart phone 1" set). With 20m cell size and PVS sky limit of 100m, the total amount of such cells is approximately 50.000. The PVS calculation currently produces approximately 10-50MB of raw data in a single file. The clustering stage compresses this into 3-9MB. Higher resolution textures could be used for mobile devices, but in practice, the low screen resolution would not allow much improvement. The PVS calculation takes more than 99% of the preprocessing time. The PVS software uses OpenGL, so good graphics cards would speed up the process. A decent desktop computer such as Athlon 3500+ with a 6th generation graphics card (NVidia 6800GT, for example) can process approximately 1 cell per second, creating a smart phone optimized model in less than a day, and a desktop optimized model in a weekend. The PVS calculation can be shared over a network. The PVS preprocess efficiency will be improved.

Device	Laptop	PDA/SP 1	SP 2
View distance	800	500	300
Texture max res.	512x512	256x256	128x128
HVS	1 pixel	1 pixel	1 pixel
PVS Resolution	768x768	320x320	208x208
VRML file size	14.3MB	14.3MB	14.3MB
Binary model size	714kB	714kB	714kB
PVS size	50MB	20MB	10MB
Clustered PVS	9MB	5MB	3MB
JPG textures	8.5MB	5MB	2MB

Table 1: 3D model preprocess settings and statistics (SP = smart phone).

PVS efficiency The efficiency of the PVS solution depends on the cell location and the surrounding geometry. The PVS solution is most efficient at street level, when the view cell is under the roof level. In such a case, 90-95% of meshes are typically culled away. In a crossing, 80-90% are culled. Depending on viewing direction, frustum culling removes another 50-80% of meshes. Above roofs, the PVS typically removes 50% of meshes, and frustum culling another 65-80%. Figure 7 presents a typical good situation at street level: almost the entire city behind the closest buildings is culled away by the PVS. Only a few visible meshes remain.

Meshes/position	Total	In PVS	In frustum	Culled%
Street	1029	40-100	20-60	95-98%
Crossing	1029	120-160	40-80	93-96%
Rooftops	1029	400-500	150-250	86-77%
Sky	1029	500-650	200-300	82-72%
Sky, looking down	1029	500-650	50-100	90-95%

Table 2: Culling efficiency statistics (a 90° field of view and 4:3 aspect ratio).

Rendering speed The approximate rendering speeds using the same preprocessed data, the "Smart phone set 1", are given in table 3. The given values are real experienced frame rates, including texture and mesh loading. Rendering speeds for the n-Gage with "Smart phone set 2" are similar as for 6630 with "Smart phone set 1". For the 6630, the higher frame rates would apply, given more memory for texture cache. Currently, 25% of the time is spent in opening texture files. The speeds are not accurate due to granularity of the system time functions. The 6630 timer has 16ms accuracy, and in the laptop environment the accuracy is 1ms. With the 6630, fastest reported rendering times are near 30-50ms, while the laptop can render the 1024x768 scenes in 2-3ms. In the laptop, we render an additional skybox to provide a nicer visual impact. Rendering the skybox affects the given numbers by 1ms.

In general, interactive rates are achieved for all platforms in all situations. At street level, the effect of the PVS is most notable. For the situation where the viewer is at sky, looking down, frustum culling dominates the culling processes (see table 2).

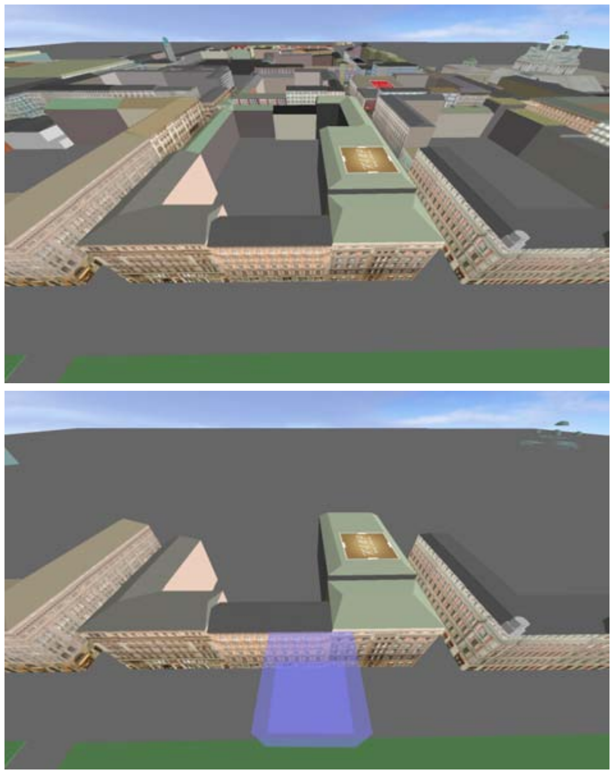


Figure 7: Visibility culling. The potentially visible set of the blue cell (below) contains only 23 out of the 264 meshes visible to the virtual camera (up).

7.2 User studies and UI development

The real results from our work on 3D maps come from user studies. Our goal was to create a platform that can be used to test various user scenarios. Before further developing the user interface, we conducted an initial field experiment after the first phase of modeling, without database integration, routing, GPS assistance or advanced GUI features. The users were given freedom to roam the entire space with the limited input controls of a PDA device. There were no special shortcuts for looking at targets or following a route. The main goal for the experiment was to research the way users orient themselves in the world, and what strategies people use for the mapping between the 3D view and the real world. We expected the results to help us to develop a more easy-to-use navigation interface. Two kinds of tasks were given: proximal mapping and remote navigation, using a 2D view from high above, and a 3D view. The results and methodologies used in the preliminary field experiment are reported in [Oulasvirta et al. 2005].

As results from the field experiment, we were able to verify several aspects of the m-LOMA system. The environment was found easy to recognize from the 3D view. People used various salient features of the environment to orient themselves, such as building façades, colors, sizes, shapes, and landmarks. When the real environment matched the 3D view, orientation and recognition was fast. On a few occasions, the unfinished 3D model was not accurate. For example, in a case where an entire façade was left untextured, the poor user who had decided to use that as a salient feature spent a lot of time searching for it in the 3D map. On another occasion, the overall texture color of a building was not accurate, and again the orientation strategy of the user failed. Because the target points were not marked in the GUI, people often lost them in the navigation tasks. The 2D view had no street names, so users were not able to locate themselves well from the 2D view, unless clear salient features were present. Sometimes users dove to street level to be able to recognize the nearby buildings. Many users expected even faster rendering. For example, diving to street level should have been almost immediate.

Platform	Dell M60	Dell X30	Nokia 6630
Resolution	1024x768	240x320	176x208
Startup time	<5seconds	5 seconds	16 seconds
Street	300-600fps	20-25fps	10-20fps
Crossing	200-400fps	15-20fps	8-15fps
At rooftops	200-300fps	10-15fps	5-12fps
In sky	150-250fps	8-15fps	4-10fps
In sky, looking down	200-300fps	10-16fps	6-14fps

Table 3: Approximate rendering speeds with the same preprocessed data set (“PDA/Smart phone 1” set) for typical platforms.

8 Conclusions and future

We have demonstrated that with a combination of preprocessing, suitable modeling methodologies and real-time rendering optimizations, a photorealistic VRML model can be turned into an efficient real-time 3D map running on a mobile device without hardware acceleration. The development of the m-LOMA system has been dominated by the use of high- and low-level 3D optimization algorithms. All our implementations have been suited to the needs of optimization. Of the various optimization methods, two are critical: visibility calculations and texture management. The visibility calculations drastically reduce the amount of primitives to render, and texture management allows us to use a model with hundreds of high-resolution textures with a decent memory consumption. Adapting 3D modeling methodologies to the selected optimization algorithms facilitates the efficiency of the algorithms. Using lightweight models makes it possible to render a large amount of buildings, allowing a large view distance.

The results from the initial field experiment indicate that the requirements for a realistic 3D map are high. When users learn to trust the 3D view, randomly appearing visual flaws are unacceptable. The street level is especially difficult: even though a careful texturing would be made, the contents of the windows in the real world change continuously. Users should be warned of this problem so that they would not rely on the street level textures for their navigation. Or, the street level textures should be cleaned of details or filled only with company logos. On the other hand, the realistic buildings are easy to recognize. Certain simplifications are allowed to the 3D models that do not seem to increase the cognitive distance between a real world and the model. For example, we have modeled all statues using billboards, which means they can only be observed from a fixed angle. Still, no complaints about the fixed viewing angle were reported in our experiment. On the contrary, all statues were easily recognized. The field experiments also demonstrated the stability of the software: during one or two hours of consecutive use, software restarts were not necessary.

Following our initial field experiment, several enhancements have been designed and implemented for the user interface. Currently, the UI features a municipally administered, manually drawn raster map, a real-time rendered vector-based 2D street map, GPS tracking using a BlueTooth GPS, routing within the map area with a route visualization and an arrow pointing at a current target, a track system to allow easier, discretized navigation at street level, etc. Content databases are being integrated to provide the users information on local sights, restaurants and public transportation. A messaging system is being implemented, allowing users to attach their own messages to the 3D scene in a common web forum fashion, allowing location-dependent public discussions. Message and information filtering is being implemented at the server side. Several other features have already been designed and are being implemented. These features will be studied iteratively in near future field experiments to finalize the m-LOMA system into a usable application.

In future, in order for mobile 3D maps to be commonly used for navigation, efficient and possibly automatic model acquisition systems should be further developed [Howard et al. 2004], where the outcome should be a lightweight, recognizable 3D model. A preprocess pipeline should be developed that would accept a range of 3D models, able to split and merge them automatically according to visibility calculations. To further standardize the process, support for visibility lists could be added to existing 3D standards such as VRML and X3D. The very interesting paper [Marvie and Bouatouch 2004] provides a framework for such an extension. If this kind of an approach could be embedded into any standard 3D model format such as VRML, X3D or M3G in an efficient manner, including support for texture management, direct rendering of such models might become feasible on mobile devices.

9 Acknowledgements

The author wishes to thank Ville Helin, the lead 3D programmer of the project. Thanks also to Petteri Torvinen for 3D modeling, Heikki Vuolteenaho for Symbian programming, Ilpo Ruotsalainen for back end programming and Antti Kantee for keeping them busy. Our implementation of the OpenGL ES was provided by Hybrid Graphics. This work has been supported by the EU Interreg IIIA.

References

ABOWD, D., ATKESON, C., HONG, J., DS. LONG, AND PINKERTON, M. 1996. Cyberguide: A mobile context-aware tour guide. *Wireless Networks 3*, 5, 421–433.

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

ANDÚJAR, C., NAVAZO, I., AND BRUNET, P. 2000. Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum 19*, 3, 499–506.

BARKOWSKY, T., AND FREKSA, C. 1997. Cognitive requirements on making and interpreting maps. In *COSIT '97: Proceedings of the International Conference on Spatial Information Theory*, Springer-Verlag, London, UK, COSIT, 347–361.

BAUS, J., CHEVERST, K., AND KRAY, C., 2005. A survey of map-based mobile guides. In Map-based mobile services - Theories, Methods and Implementations, 197-213.

BESSA, M., COELHO, A., AND CHALMERS, A. 2004. Alternate feature location for rapid navigation using a 3d map on a mobile device. In *MUM 2004 XXX*, ACM.

BLESCHMIED, H., ETZ, M., AND HAIST, J. 2005. Providing of dynamic three-dimensional city models in location-based services. In *MOBILE MAPS 2005 - Interactivity and Usability of Map-based Mobile Services. A workshop.*, Mobile HCI.

BORNTRÄGER, C., CHEVERST, K., DAVIES, N., DIX, A., FRIDAY, A., AND SEITZ, J. 2003. Experiments with multimodal interfaces in a context-aware city guide. In *Proceedings of Mobile HCI 2003*, Mobile HCI.

BOUVILLE, C., MARCHAL, I., AND BOUGET, L. 2005. Efficient compression of visibility sets. In *International Symposium on Visual Computing (ISVC 2005)*, Sprnger-Verlag, ISVC.

BRACHTL, M., SLAJS, J., AND SLAVÍK, P. 2001. Pda based navigation system for a 3d environment. *Computers and Graphics 25*, 4, 627–634.

BURIGAT, S., AND CHITTARO, L. 2005. Location-aware visualization of vrml models in gps-based mobile guides. In *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*, ACM Press, New York, NY, USA, ACM, 57–64.

CHEVERST, K., DAVIES, N., MITCHELL, K., FRIDAY, A., AND EFSTRATIOU, C. 2000. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, ACM, 17–24.

CHHUGANI, J., PURNOMO, B., KRISHNAN, S., COHEN, J., VENKATASUBRAMANIAN, S., AND JOHNSON, D. S. 2005. vlod: High-fidelity walkthrough of large virtual environments. *IEEE Transactions on Visualization and Computer Graphics 11*, 1, 35–47. Member-Subodh Kumar.

CLARK, J. H. 1976. Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10, 547–554.

DARKEN, R. P., AND SIBERT, J. L. 1993. A toolset for navigation in virtual environments. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, ACM Press, New York, NY, USA, 157–165.

FATHAMMER, 2005. Fathammer x-forge. http://www.fathammer.com (last accessed, December 2005).

FEINER, S., MACINTYRE, B., HÖLLERER, T., AND WEBSTER, A. 1997. A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. In *ISWC '97: Proceedings of the 1st IEEE International Symposium on Wearable Computers*, IEEE Computer Society, Washington, DC, USA, IEEE, 74.

FUCHS, H., KEDEM, Z., AND NAYLOR, B. 1980. On visible surface generation by a priori tree structures. *Computer Graphics (Proc.SIGGRAPH'80) 14*, 3, 124–133.

HESINA, G., AND SCHMALSTIEG, D. 1998. A network architecture for remote rendering. In *DIS-RT '98: Proceedings of the Second International Workshop on Distributed Interactive Simulation and Real-Time Applications*, IEEE Computer Society, Washington, DC, USA, 88.

HOWARD, A., WOLF, D. F., AND SUKHATME, G. S. 2004. Towards 3d mapping in large urban environments. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE.

IACUCCI, G., KELA, J., AND PEHONEN, P. 2004. Computational support to record and re-experience visits. *Personal and Ubiquitous Computing 8*, 2, 100–109.

JCP, 2005. Jsr 184: Mobile 3d graphics api for j2me. http://www.jcp.org/en/jsr/detail?id=184 (last accessed, December 2005).

KHRONOSGROUP, 2005. Opengl es - the standard for embedded accelerated 3d graphics. http://www.khronos.org/opengles (last accessed, December 2005).

KRAY, C., ELTING, C., LAAKSO, K., AND COORS, V. 2003. Presenting route instructions on mobile devices. In *IUI'03*, 209–224.

KRÜGER, A., BUTZ, A., MÖLLER, C., STAHL, C., WASINGER, R., STEINBERG, K.-E., AND DIRSCHL, A. 2004. The connected user interface: realizing a personal situated navigation service. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, ACM Press, New York, NY, USA, ACM, 161–168.

LINTURI, R., KOIVUNEN, M.-R., AND SULKANEN, J. 2000. Helsinki arena 2000 - augmenting a real city to a virtual one. In *Digital Cities, Technologies, Experiences, and Future Perspectives [the book is based on an international symposium held in Kyoto, Japan, in September 1999*, Springer-Verlag, London, UK, 83–96.

LUEBKE, D., REDDY, M., COHEN, J., VARSHNEY, A., WATSON, B., AND HUEBNER, R. 2005. *Level of Detail for 3D Graphics*. Morgan Kaufmann.

LYNCH, K. 1960. *The Image of the City*. Cambridge: M.I.T.Press.

MARVIE, J.-E., AND BOUATOUCH, K. 2004. A vrml97-x3d extension for massive scenery management in virtual worlds. In *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*, ACM Press, New York, NY, USA, ACM, 145–153.

MIETTINEN, V., 2005. Building scalable 3d applications. ACM SIGGRAPH 2005 Course #35: Developing Mobile 3D Applications With OpenGL ES and M3G, August.

MOLLER, T., AND HAINES, E. 1999. *Real-time rendering*. A. K. Peters, Ltd., Natick, MA, USA.

NIRENSTEIN, S., BLAKE, E., AND GAIN, J. 2002. Exact from-region visibility culling. In *Proceedings Eurographics Rendering Workshop*, S. Gibson and P. Debevec, Eds., Eurographics, 191–202.

OULASVIRTA, A., NIVALA, A.-M., TIKKA, V., LIIKKANEN, L., AND NURMINEN, A. 2005. Understanding users' strategies with mobile maps. In *Mobile Maps 2005 - Interactivity and Usability of Map-based Mobile Services, a workshop*, Mobile HCI.

OULASVIRTA, A. 2005. The fragmentation of attention in mobile interaction, and what to do with it. *interactions 12*, 6, 16–18.

PAAY, J., AND KJELDSKOV, J. 2004. Understanding and modelling built environments for mobile guide interface design. *Behaviour and Information Technology 24*, 1, 21–35.

PARALLELGRAPHICS, 2005. Pocket cortona - mobilize your enterprise! http://www.parallelgraphics.com/products/cortonace/ (last accessed, December 2005).

PASMAN, W., AND JANSEN, F. W. 2003. Comparing simplification and image-based techniques for 3d client-server rendering systems. *IEEE Transactions on Visualization and Computer Graphics 9*, 2, 226–240.

POSPICHIL, G., UMLAUFT, M., AND MICHLMAYR, E. 2002. Designing lol@, a mobile tourist guide for umts. In *Proceedings of Mobile HCI 2002*, Springer-Verlag, Mobile HCI, 140–154.

RAKKOLAINEN, I., TIMMERHEID, J., AND VAINIO, T. 2001. A 3d city info for mobile users. *Computers and Graphics 25*, 4, 619–625.

REDDY, M., LECLERC, Y., IVERSON, L., AND BLETTER, N. 1999. Terravision ii: Visualizing massive terrain databases in vrml. *IEEE Comput. Graph. Appl. 19*, 2, 30–38.

SCHMALSTIEG, D., AND GERVAUTZ, M. 1996. Demand-driven geometry transmission for distributed virtual environments. *Computer Graphics Forum 15*, 3, 421–431.

SMITH, S. P., AND MARSH, T. 2004. Evaluating design guidelines for reducing user disorientation in a desktop virtual environment. *Virtual Reality 8*, 1, 55–62.

SUDARSKY, O., AND GOTSMAN, C. 1997. Output-senstitive rendering and communication in dynamic virtual environments. In *VRST '97: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM Press, New York, NY, USA, ACM, 217–223.

TELLER, S. J. 1992. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, University of California at Berkeley.

THIBAULT, W. C., AND NAYLOR, B. F. 1987. Set operations on polyhedra using binary space partitioning trees. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, ACM, 153–162.

VAINIO, T., KOTALA, O., RAKKOLAINEN, I., AND KUPILA, H. 2002. Towards scalable user interfaces in 3d city information systems. In *Mobile HCI '02: Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*, Springer-Verlag, London, UK, Mobile HCI, 354–358.

VAN DE PANNE, M., AND STEWART, J. 1999. Efficient compression techniques for precomputed visibility. In *Rendering Techniques '99 (Proceedings of the 10th EG Workshop on Rendering*, Eurographics Association, Springer Computer Science, Eurographics, 305–316.

VINSON, N. G. 1999. Design guidelines for landmarks to support navigation in virtual environments. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, ACM, 278–285.

WILLIAMS, L. 1983. Pyramidal parametrics. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, ACM, 1–11.