Jiri Zara
Czech Technical University in Prague
Karlovo Nam. 13
121 35 Praha 2
Czech Republic

Web-Based Historical City Walks: Advances and Bottlenecks

Abstract

This paper deals with a set of useful methods for presenting large-scale models of real cities in a web environment. While acquisition issues are outlined only briefly, this study focuses on the optimal organization of city models and efficient rendering techniques. We also address data optimization for real-time delivery. The methods under discussion are illustrated by examples taken from several existing web presentations developed by our students, especially from the Virtual Old Prague project. Although the principles are general, specific attention is paid to the use of standards developed by the Web3D Consortium.

1 Introduction

Visualization of existing cities is a problem that bridges two different research areas—Computer Vision and Computer Graphics. Due to the complexity and diversity of cities around the world, especially historical cities, it is a challenging task to develop efficient acquisition, storing, transfer, and presentation algorithms. In this paper, we concentrate on a specific area—web-based presentation. We focus on a smooth walk through a large-scale 3D urban environment in real time, using standard technologies (Virtual Reality Modeling Language). Limitations due to the web environment are discussed, and ways to overcome them are shown in Section 2. Section 3 introduces specific features of web-based 3D graphics applications. Data structures for large city models are discussed in Section 4. Implementation issues on the client side are presented in Section 5. Open problems are listed in Section 6.

2 Related Work

Currently, much attention is being paid to the automatic *reconstruction* of city models. Since we have to model existing inhabited areas, various approaches utilize photographs. The general city layout is usually reconstructed using aerial images, possibly combined with GIS (Geographic Information System) data (Collins et al., 1998; Kunii & Chikatsu, 2003). This typically results in a model of a terrain covered by houses with a block geometry and roofs of simple shape (Moons, Frere, Vandekerckhove, & Van Gool, 1998). Planar

Presence, *Vol. 15, No. 3, June 2006,* 262–277
© 2006 by the Massachusetts Institute of Technology

*Correspondence to zara@fel.cvut.cz

areas such as streets and simple roofs can be covered by textures obtained directly from aerial images, while the reconstruction of a 3D facade requires an additional and generally a much greater effort in terms of acquisition and processing time. Several technical approaches can be identified, including a non-calibrated camera (Koch, Pollefeys, & Van Gool, 2000), stereo photography, continuous video recording (Zheng & Tsuji, 1998; Zheng & Shi, 2003), and a laser scanner in combination with GPS (Global Positioning System) devices. A representative example is the MIT City Scanning Project (Coorg & Teller, 1999), which enables automatic modeling of a large urban environment.

Models obtained from images often belong to one of two extremes. The buildings are either constructed from simple planar (textured) facades only, or they consist of a large number of unorganized triangular meshes. The optimal model complexity for real-time presentations lies somewhere between these extremes, that is, individual houses should contain from tens to hundreds of textured polygons, preferably structured into levels of detail (LOD). The creation of such models requires slower, interactive work to find geometrical architectural elements (Taylor, Debevec, & Malik, 1996). The pioneering commercial software was Canoma by MetaCreations, unfortunately no longer supported. A promising approach for future studies seems to be knowledge-based reconstruction from images, where typical elements such as windows, balconies or arcs can be efficiently found and marked (Zlatanova & van den Heuvel, 2002).

To conclude the overview of 3D reconstruction techniques, a curious issue concerning visual obstacles should be mentioned here. Obstacles such as trees, street lamps, traffic lights, cars, and people represent a fundamental complication for many reconstruction algorithms. Such objects have to be removed (automatically or manually) in the reconstruction phase, but finally they should be added back to the model to improve the natural look of the city.

While 3D reconstruction of large urban sites is still a subject of a research, *real-time visualization* of 3D models is much more elaborated upon (Slater, Steed, & Chrysanthou, 2002). The most useful principles include the utilization of levels of details (both in the geometrical and in the texturing meaning), culling, visibility preprocessing (either general, see Durand, 2000, or specific to the urban environment, see Bittner, Wonka, & Wimmer, 2001), and impostors (Sillion, Drettakis, & Bodelet, 1997; Decoret, Schaufler, Sillion, & Dorsey, 1999) replacing more distant objects and a complex background. Even for a distributed environment like the web, many computer graphics algorithms have been created/adopted, for example, mesh streaming (Hoppe, 1996). On the other hand, web-based graphics is still considered as a tool for presentation of single objects (e.g., in e-commerce), and thus real-time visualization of large-scale city models has not yet been fully solved. The traditional trade-off between speed and quality of rendering is complicated by the fact that the model is stored in a distant computer—a web server.

Two main approaches are currently used for city presentations on the web—image based and model based rendering. *Image based methods* directly utilize photographs, thus achieving a high quality of rendering. A typical example is QuickTime VR technology, where cylindrical panoramic views represent the virtual environment around a user. It is suitable for open areas like town squares, hills, and so forth. Narrow and curved streets, typical for medieval cities, do not fit well into the cylindrical shape of a panorama, thus a sequence of several neighboring panoramas with a small radius has to be used. An interesting improvement suggested by Zhyeang and Shi (2003) is called *route panorama*. It is based on the creation of two linear panoramas, one for the left view and one for the right view when walking along a street. A special viewer merges these two panoramas, using perspective projection, into a single image. Background images at both ends of a street can also be included. Regardless of the high speed and quality of panoramic images, dynamic and interactive behavior is limited to hyperlinks associated to sensitive areas only. It is difficult to include animated objects like trams or people in panoramic views.

The second approach is the classical *rendering of 3D models*. Virtual Reality Modeling Language (VRML) is a representative technology designed especially for the web. Although several competitive solutions exist, they

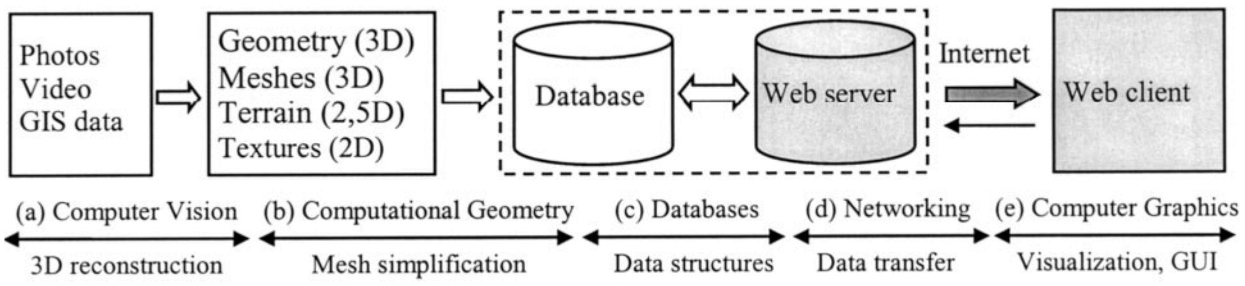


Figure 1. *A general scheme of the processing of city data. Various areas of computer science participate in this complex but challenging web application. While the data flow in the left part is unidirectional, interactive visualization requires two-way data transfer.*

are either limited to a certain computing platform (e.g., SVR technology by Superscape) or they are designed for presentation of an individual object (e.g., MetaStream technology by Viewpoint) or specialized to an excessively narrow geographical application area (e.g., DILAS by GEONOVA). Due to the universality and complex functionality of VRML (collision detection, terrain following, simulation based on event processing, scripting with the use of external programming languages, etc.), VRML browsers are generally considered to be very slow in comparison with, for example, game engines. This drawback will persist into the future, although the Web3D Consortium has prepared a new version of a language called X3D, which allows a layered design and application specific profiles. Real-time navigation in a well-modeled and richly-textured virtual city environment will always require many computing resources and much data optimization.

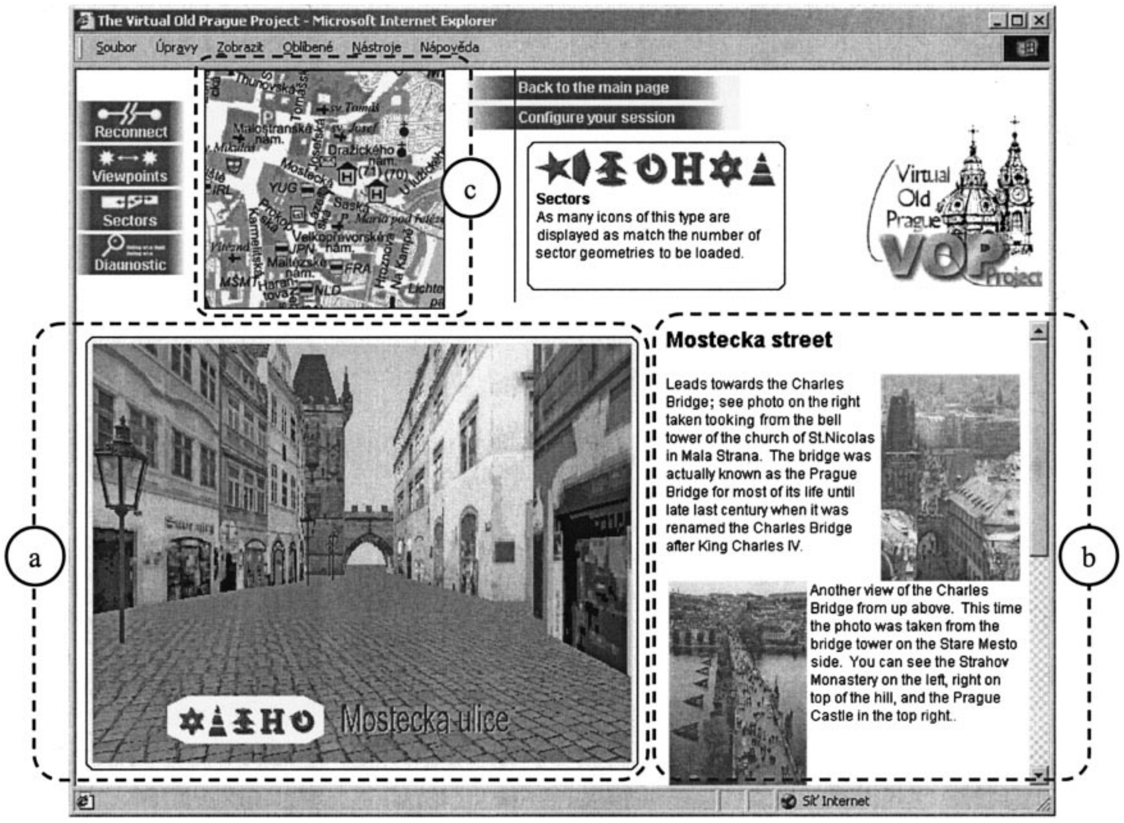
The most difficult goal for a web city presentation is a *smooth walk* through a large urban area. Most existing web cities consist either of one big unstructured model (which is hard to download quickly and render efficiently), or of several independent smaller models connected by hyperlinks only. When users move from one such a part of a city to another part, newly downloaded models usually fully replace the current 3D data like a new HTML document replaces the previous one after a hyperlink has been activated. This is more like teleportation of the user, rather than continuous movement in a specific direction. While this behavior is naturally acceptable for text documents, it is undesirable for the illusion of a walk through a 3D environment. A seamless visual presentation can be better achieved by continuous processing of geometrical models rather than by a panorama. The rest of the paper thus only concentrates ways of rendering spatial geometrical models.

3 Web-Based Visualization

The main parts for processing urban data are shown in Figure 1. The difference between single computer rendering and web-based visualization is depicted using the filled areas in the figure. In a web environment, the following three components play specific roles:

1. **Web Server.** The server is responsible for storing data and transferring it to clients. Since it serves a large number of web users, it does not provide complex computations for individuals, but, instead, simple tasks like selecting a piece of data from a larger model, converting it to a specific format (e.g., VRML), packing, and sending out. A server typically does not hold any state information about clients, thus all communication between a client and a server has to be designed as stateless.

2. **Web Client.** The client deals with rendering and interaction with a user. Due to the complexity of city data, it does not render the whole model, but only the neighboring area around the user's avatar. As the avatar moves in 3D space and interacts


The Virtual Old Prague Project - Microsoft Internet Explorer
Soubor Úpravy Zobrazit Oblíbené Nástroje Nápověda
Reconnect
Viewpoints
Sectors
Diagnostic
Back to the main page
Configure your session
Virtual Old Prague VOP Project
Mostecka ulice
Síť Internet
a
b
c

Sectors
As many icons of this type are displayed as match the number of sector geometries to be loaded.

Mostecka street

Leads towards the Charles Bridge; see photo on the right taken looking from the bell tower of the church of St.Nicolas in Mala Strana. The bridge was actually known as the Prague Bridge for most of its life until late last century when it was renamed the Charles Bridge after King Charles IV.

Another view of the Charles Bridge from up above. This time the photo was taken from the bridge tower on the Stare Mesto side. You can see the Strahov Monastery on the left, right on top of the hill, and the Prague Castle in the top right..

Figure 2. *A web page with a virtual city can contain several interactive components allowing synchronized multimedia presentation. The example taken from the Virtual Old Prague project shows the three most important parts. (a) A 3D scene window. (b) An HTML document. (c) A 2D navigation map.*

with the model, the client generates requests for new data from the server. Incoming (asynchronous) data has to be recognized and seamlessly added to the current presentation, thus creating the illusion of a smooth walk through a whole city in real time.

3. **Network.** Since the speed of data transfer varies from tens of Kbits (phone lines) up to hundreds of Mbits, it is difficult to tune the data size to a specific connection speed. Web-based city models should be initially optimized for low speed connection, with the possibility of enlarging the data sizes (geometrical details, textures) for higher speed connection. Here the LOD principle is very useful, not only for optimizing the rendering, but also for transferring the data. The web also enables data transfer from a client to a server. This is typically used for sending simple data requests only, and not for rich interactive work. Due to the time delay that is characteristic of the network, interactions between a user and a virtual city environment have to be performed primarily on the client side.

Although our focus is on a 3D virtual environment for city presentations, other ways to show a city on the web should also be mentioned here. Since many users are not familiar with 3D navigation, common web techniques based on hyperlinks and/or sensitive 2D maps are also useful. Figure 2 shows one such approach, currently available at http://www.cgg.cvut.cz/vsp/, which uses a standard VRML technology for 3D data, combined with a Java applet for synchronized movement of a 2D avatar icon on a city map.

We should mention that only experts from the 3D area and game players are able to navigate freely in 3D space. We have been surprised to find that most users do not expect to be able to walk through a city by dragging a mouse in a particular direction. Instead, they think of a 3D rendering window as an output only screen where some animation can be seen. Such users are able to control their virtual movement by pressing buttons or moving sliders, rather than to directly navigate through a 3D browser window. Although these problems are beyond the scope of this paper, some support for old fashioned users should be taken into consideration when designing a city web presentation. This includes precalculated walks (guided tours), a sequence of simply accessible viewpoints, sensitive areas for the click-and-go navigation paradigm, or a search function providing animated movement to a target place in a city.

4 Data Structures for City Models

This section deals with data structures suitable for describing a city and parts of a city. Due to the complexity of a city environment, more than one data structure has to be used. On a macroscopic level, a city is subdivided into smaller blocks with a view to efficient, nonredundant data transfer through the web. Such city subdivision techniques are described in Section 4.1. We further concentrate on the most recognizable city objects—houses. They require special care in terms of geometric description, texturing, and the level of detail utilization. These issues are introduced in Section 4.2.

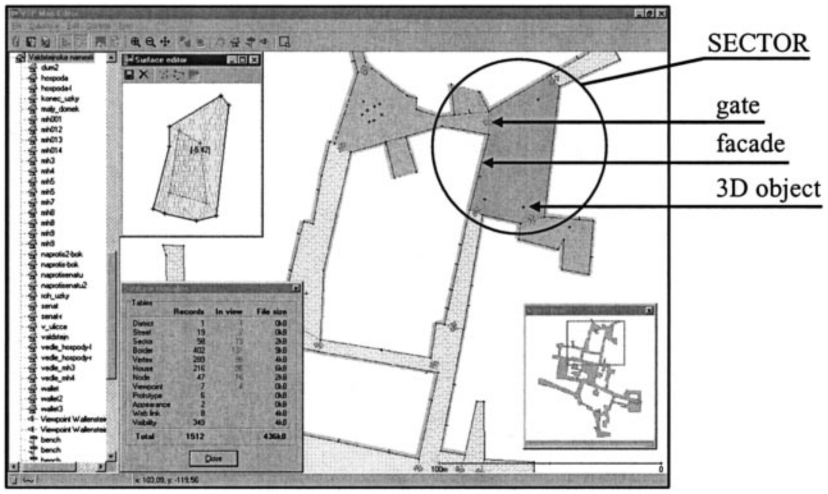
4.1 Subdividing City Spaces

Generally, cities have a hierarchical structure, headed by districts (quarters) and continuing with smaller elements like streets and then individual houses. In addition to this logical tree arrangement, topological information about street adjacency can be built in the form of a planar graph. Other auxiliary data structures coming from the area of Computer Graphics help to increase the speed of rendering, utilizing k-d trees, octrees, grids, bounding volume hierarchies, and so forth. Individual visible parts of a city (geometry, textures) are thus always extended by invisible additional structures with a global character.

Web specific limitations influence the arrangement of city data. When the city is to be large-scale, the initial transfer of a global auxiliary data structure causes a big delay before seeing any image on the client side. Moreover, users do not need to download the whole logical structure of a city when enjoying a virtual walk around one square only. The key issue is how to subdivide both the 3D model and the corresponding auxiliary speeding-up data structures into smaller data packages suitable for transfer from server to client. Such data packages should be of limited size, self-contained, and have unique identification. A client should be able to insert them easily into already downloaded parts and connect them with other existing data structures.

A special requirement concerns visibility. In many spatial applications, some kind of visibility is precomputed, thus minimizing the data to be transferred and rendered for given users' position(s). From-area visibility (called eye-to-cell visibility in Slater, Steed, & Chrysanthou, 2002) is very suitable for a city environment. For every small region, a potentially visible set (PVS) is computed and stored. The data packages should correspond with the cells contained in the PVS sets.

From the programmer's point of view, a grid seems to be an appropriate structure for subdividing a city. Since the city layout is planar, a two-dimensional grid is sufficient for most cases. When large buildings require a three-dimensional structure for processing interior parts, a three-dimensional grid, a hierarchy of grids, or a combination of a quadtree and an octree can be applied. Each cell of a regular grid is easily identifiable and accessible via indexing; connection between cells is straightforward. However, we recommend the use of a grid only as an auxiliary structure, for example, for fast identification of a user's position in a city but not as a primary subdivision technique. Few real cities are built on strict geometrical rules (Lynch, 1960). The arrangement of streets, squares, parks, rivers, and other parts



Tables

	Records	In view	File size
District	1	1	0kB
Street	19	2	0kB
Sector	58	13	2kB
Border	402	131	9kB
Vertex	289	96	4kB
House	216	56	6kB
Node	47	16	2kB
Viewpoint	7	4	0kB
Prototype	6		0kB
Appearance	2	-	0kB
Web link	8	-	4kB
Visibility	349	-	4kB
Total	1512		436kB

x: 103,09, y: -119,56

Figure 3. *The sector is the basic data structure for a virtual city. House facades and gates to other sectors enclose a polygonal ground with given elevation(s). Spatial objects such as statues and street lamps are placed inside the sector area. The snapshot has been taken from the MapEdit utility belonging to the Virtual Old Prague project.*

does not fit into any orthogonal grid, especially in historical cities. Moreover, the correlation between grid cells and PVS sets is generally low. If a grid cell contains several streets, nothing can be said about their mutual visibility. A partially visible grid cell has to be transferred as a whole. Using grid cells as basic data packages, too much redundant (because invisible) data would be sent through the net, thus delaying the transfer of other necessary visible elements.

The city subdivision technique should follow the real arrangement of the city elements. Since streets are natural and common objects when searching for city information or walking in a city, we recommend that the street be taken as the basic element for subdividing a virtual city. However, long streets with several crossroads are not suitable for visibility preprocessing, thus short street segments are used instead. We have designed a data structure called a sector that has been successfully employed in various virtual cities. The sector is tightly related to street segments, but is not limited to them. It can describe squares as well as interiors. A typical sector represents one street segment surrounded by two opposite rows of houses and closed by crossroads at both ends. While a simple crossroads will belong to one of the adjacent street sectors, large crossroads and squares are modeled by their own sectors. To keep the data size per sector within limits, long street segments or river banks are divided into a chain of sectors. A set of sectors forms a city. The internal structure of a sector is shown in Figure 3.

A sector holds the following information:

4.1.1 Unique ID. This serves to identify the user's position, and also for further connection with neighboring sectors.

4.1.2 Ground. This is the surface on which the avatar moves. It is represented by a polygonal area with a given elevation, a height field, or a triangular mesh. It can be further covered either by a simple texture or by a real geometry, such as sidewalks, stairs, tram rails, etc.

4.1.3 Border Geometry. While the ground represents horizontal information, house facades placed on the ground border constitute vertical parts of the sector. All houses belonging to a given sector are considered as a solid border geometry. A user cannot pass through them except in cases when a house door leads to an interior, that is, to another sector.

4.1.4 Gates. An edge of the ground polygon that is not occupied by a border geometry is called a gate. It holds the ID of a neighboring sector, thus enabling it to be connected via its own corresponding gate. A user can leave a given sector only through a gate. A special case is the end of the whole city model, where the gate is modeled as a solid face with a background image placed on it. Then the gate does not serve as a passage but as an obstacle. Gates are important not only for holding adjacency information among sectors but also for visibility computations. The gate represents a portal in the terminology of visibility processing (Luebke & Georges, 1995), while the sector represents a cell.

4.1.5 Standalone Objects. A sector area is usually enriched by additional spatial models. These include a variety of static objects like trees, animated objects (water fountains or moving trams), interactive objects (street lamps to be switched on/off), sensitive objects (signboards with hyperlinks leading to company web sites), and so on.

4.1.6 Viewpoint List. Predefined camera positions are stored for real-time presentation.

4.1.7 List of Visible and Important Sectors (Lvis). While gates are directly applicable for storing adjacency information, precomputed visibility results are stored in the *Lvis* list. This contains the IDs of sectors potentially visible from a given sector. A more detailed explanation follows below.

The Lvis list is a special feature that holds important information about visibility. This information is later used both for speeding up the rendering and for loading sectors in advance. To create the Lvis list, a visibility-preprocessing algorithm based on cell-to-cell evaluation has to be executed. Such algorithms mostly assume a convex shape of a cell (sector area). To meet this requirement for curved nonconvex streets, a convex hull or even a simple bounding volume of the sector is a sufficient alternative. If a user walks on the ground, and flying is not permitted, then the visibility problem can be transformed from full 3D space into 2.5D or even 2D space only.

Originally, the Lvis list was designed as an equivalent to PVS (potentially visible set). After testing practical implementation in the Virtual Old Prague project, we extended its functionality to make data reading/releasing more efficient during rendering. We noticed that users walking through a city tend to return repeatedly to already visited squares and to other sectors characterized by valuable historical/architectural/artistic contents. These special places were often removed from the client memory as users left them and passed through a narrow street. Soon, these sectors were again reloaded when users returned via another street. Although the cache of a web browser keeps the already downloaded sectors and a full download from the server is not necessary, the time for memory allocation and the insertion of repeatedly read geometry into a scene graph is considerable for real-time presentation. We decided to hold such repeatedly visited sectors in memory as long as possible. The solution was to add their IDs to the Lvis lists of all sectors in a specific vicinity. It is for this reason that Lvis stands for list of visible and important sectors.

While visibility preprocessing is an automatic task, the important sectors mentioned above have to be added to Lvis manually. We consider this attention to important sectors as a secondary activity for the fine-tuning of the overall system performance. In most cases, the Lvis list can be directly derived from the PVS resulting from a visibility algorithm.

An interesting problem is the visual appearance of gates. As they represent passages to neighboring sectors, they are not solid but transparent. A user should see city parts on a background through a gate. If such parts are not yet loaded from the server, a predefined image can be temporarily placed on a gate. This technique uses an impostor and utilizes the idea of image-based rendering in combination with classical rendering methods. To make an impostor, we can utilize either photos taken from a real city or images generated by the rendering of the background geometry. The first method highlights

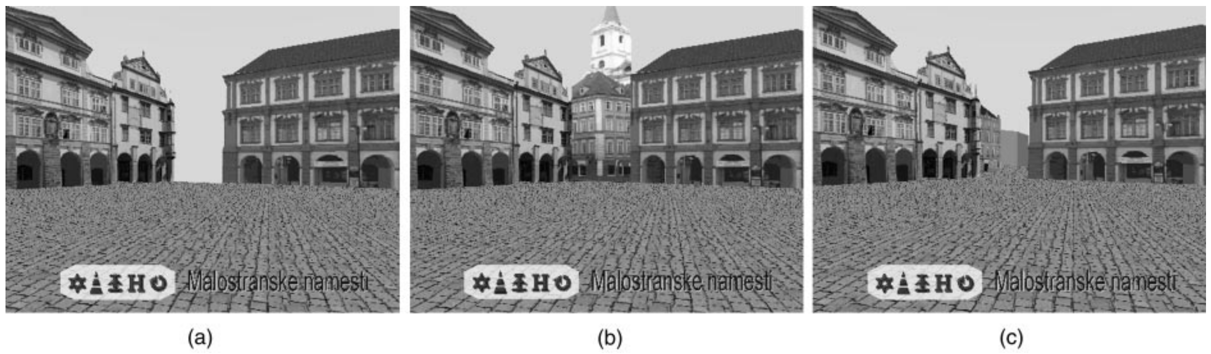


Figure 4. *Impostors temporarily placed at sector gates improve the visual look. (a) An individual sector without any impostor. (b) Impostor at a gate. (c) A neighboring sector was loaded and the original impostor removed. The house in the background has no texture. This sequence of snapshots is a rather negative example, since the impostor was created using a photo taken from an inappropriate position. Big differences and even quality degradation can be seen when snapshots (b) and (c) are compared. Impostors should be made of images resulting from 3D models.*

differences between a photo and a model (as seen in Figure 4), but allows us to store the distant background of the environment that is not stored in a city model, for example, mountains, forests, and so forth. The second method can be executed automatically for all gates. A camera is usually placed at the most distant position within a sector and oriented toward a gate. Since the area of a sector is relatively small, visible errors like false parallaxes are not significant when walking within a sector. Moreover, neighboring sectors soon replace the nearest impostors. Newly loaded impostors belonging to neighboring sectors are farther from the current user's position, so the illusion of depth emerges. Thanks to the fine granularity of the subdivision of the city, and the temporal character of gate impostors, one simple planar face is sufficient for one impostor. This is different from the method published by Sillion et al. (1997). The authors replace the background geometry by several faces with various depths. Since their city cells are larger, the changes in the mutual positions of distant buildings are more significant during a user's walk. In fact, they create a special geometrical level of detail replacing the background geometry.

Although impostors placed at gates solve many problems of missing, that is, not yet downloaded geometry, they have certain limitations. A sector representing an open area (a meadow in a park, a hilltop, etc.) should be surrounded by gate impostors only. Due to the polygonal shape of a sector ground, perspective errors and cracks can arise on the vertical edges of gate impostors. A panoramic background and image-based rendering would be welcome. Unfortunately, we have not met a system where a mixture of geometrical data and a panoramic background would be seamlessly merged and rendered. We would like to point out that parts of the background have to be replaced by geometrical models as the user walks in a specific direction.

Another problem arises when users want to fly over a city. Although this is definitely not a usual way for tourists do their sightseeing tours, the possibility of flying in virtual space is highly attractive. Unfortunately, a flyover requires almost all city data to be available, which is not the case for web-based city presentations. Since very large parts of a city are visible from the air, the visibility preprocessing that is suitable for streets, that is, PVS based on sectors, is not usable for this task. Instead, the LOD principle plays a more important role.

A combination of simplified house models viewed from the air and a terrain covered by multi-resolution textures can be utilized. This approach is partially supported, for example by the GeoLOD node in the GeoVRML/X3D standard specification, where the terrain is represented by a quadtree structure. In our opinion, a fast and believable flight over a virtual city using web browsers will remain a dream for many years. Current standard technologies constrain virtual visitors to be bound to the ground. On the other hand, the ever-increasing performance of graphics cards and especially a very high bandwidth allowing fast streaming of geometrical and texture data are promising conditions to enable web users to fly in a large-scale virtual environment.

In summary, we propose a city model consisting of an unordered list of sectors. This linear arrangement is very suitable for databases storing an arbitrary city description. Adjacency information can be directly obtained from the sector data via unique sector IDs. If the city model grows as time passes, new sectors can be naturally added to an existing database. Furthermore, this model is highly scaleable. The sector definition fulfills all the criteria specified at the beginning of this section. The sector is a self-contained data entity that can be individually transferred and rendered. Lvis lists allow further sectors to be loaded. Dynamic sector management is described in Section 5.1.

4.2 House Structures

Great efforts have been made in research on the automatic reconstruction of buildings, but still no handy solutions are available. Triangular meshes generated from images are too large and unorganized, while web visualization requires almost the opposite kind of data—small and well structured. Most virtual cities currently available on the web were not automatically reconstructed, but were created manually or semi-automatically. Models of houses are typically converted from CAD drawings (if available) or created using programs for modeling. A widely used commercial modeler is the 3D Studio Max, from which the data can be exported to various formats. To ensure a certain structure for a house, a specialized program is better than a general modeler. The following text explains the advantage of specialized tools, particularly when a model has to be transferred through the web in several levels of detail (LOD).

Most methods for generating LODs take one initial mesh as an input and create simplified meshes upon various decimation criteria. This works well for a single complex object like a statue or a highly detailed relief facade. On the other hand, it fails for the case of a city consisting of hundreds of houses, that is, hundreds of potential meshes. A common simplification of several houses would cause visible errors due to facades and roofs merging into nonexisting faces. Simplification of individual buildings with a low computational effort is required. Instead of progressive LOD representations, several discrete models per house suffice.

We have proposed a four-step LOD representation called Urban LOD (Zara et al., 2001). It is based on the idea of visual importance. Optimization of the data flow also takes into account human perception. Two most recognizable features of a house have been identified—an outline of the facade, and the windows/doors. A combination of geometry and textures is then arranged into four discrete LODs. This principle has been successfully used in the Virtual Old Prague project. Based on our experience, we have slightly changed the third LOD representation and redesigned the fourth level. While the original proposal assumed an arbitrary 3D model for the fourth level, we prefer a model that is semi automatically created from previous LODs. The result is shown in Figure 5.

The simplest model consists of a few faces (facades and roofs) filled by solid colors. This is similar to block models automatically reconstructed from aerial images or GIS systems. A house silhouette is the main feature visible in this model. The second LOD representation concentrates on visually important parts of a facade. These are usually windows, doors, arcades, or frescos. They can be directly added to the previous LOD in the form of textured polygons placed in front of a facade. As these architectural/artistic objects are often repeatedly arranged along a house, the same texture can be duplicated (translated and even scaled), thus saving both time for transfer and the size of the texture memory in a client computer. We want to stress that a texture size of 64×64 pixels is fully sufficient here, and 256 colors is usually enough. Such a compressed texture file with a palette (PNG in our case) does not exceed 3 kB in size.

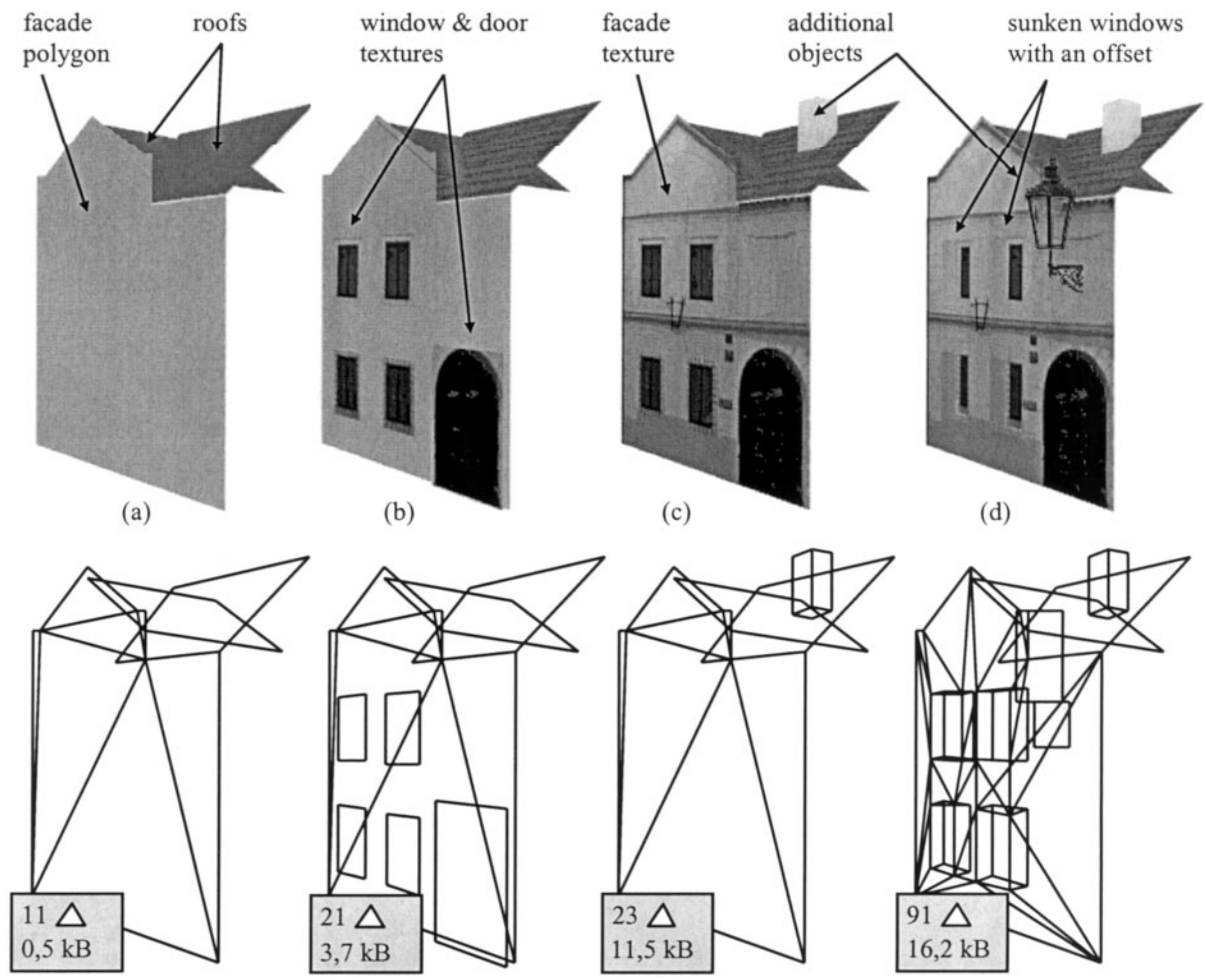


Figure 5. *The LOD principle can utilize the architectural structure of a house. The upper row shows four discrete LOD representations, while the lower row contains the corresponding wire-frame models and data sizes. Note that the window textures already transferred from the server for level (b) are used again for level (d).*

The third LOD was designed with the aim to further increase the visual quality while holding the overall number of polygons unchanged. A facade is covered by one larger texture in this representation. Since the smaller polygons added to the facade in the second LOD have been removed, other geometrical objects such as chimneys or balconies can be used instead. Finally, the last LOD is composed from the geometrical information from the second and third LODs. Windows (and possibly other visually important objects) are inserted into a facade using a given depth. The final facade gets a real 3D appearance.

Our approach requires interactive work when preparing house models, but automatic preprocessing based on knowledge-based recognition seems to be possible. The currently used utility program takes an orthophotograph of the facade as an input and allows the user to interactively define the polygonal outline, the shape and slope of the roof, the solid fill colors for the facade and roof, visually important textures and their repetitive positions, and insertion depths for windows/doors. Then four urban LODs are created.

Unlike mesh simplification and streaming algorithms, urban LOD representations are not created incrementally. Information included in the second level is not immediately used in the following level, but later in the fourth level. This is due to the effort to efficiently transfer a mixture of geometrical and texture information from the server to the client and to present the most visually important data as soon as possible. Sample data sizes per individual LOD are shown in Figure 5. The amount of transferred data increases evenly as the complexity of the LOD increases. Pop-up effects that can be seen when switching among discrete levels are acceptable to web users. This is similar to the situation when a web page is progressively downloaded and the images and larger tables are displayed step by step. These effects can be minimized by blending between LODs and also by increasing the distances over which the LODs are switched. The use of discrete LODs minimizes computations on the client side, since the only continuously updated value is the avatar-to-object distance.

From the practical point of view, even an incomplete city model can be presented on the web in an early stage of virtual city construction, for example, one planar facade with one texture per house only (third urban LOD). Other LODs can be modeled later. Visitors to virtual cities are impressed when they are fully surrounded by many different house facades forming the city, while the architectural decor plays only a secondary role.

To conclude the discussion of house models, attention should be paid to the preparation of textures. Again, the limited capacity of the web influences the size of the textures used. The memory architecture of graphics cards on the client side also plays an important role. Texture memory is usually allocated in blocks of constant size. The texture image resolution should be a multiple of this block size, otherwise the texture memory is fragmented and not fully utilized. Images with sizes corresponding to a power of 2 (e.g., 256×256, 128×512, etc.) are generally recommended. A reasonable upper limit for one dimension is 512, exceptionally 1024. It is important to point out that one true color image with resolution 1024×1024 requires 3 MB of texture memory after decompression. When using mipmapping, an additional 1 MB is required. Ten such textures would easily overload a medium quality graphics card with 32 MB texture memory. Image compression (e.g., JPEG) is important for data transfer but not for standard graphics cards.

To speed up the transfer of several small textures belonging to a single house (e.g., roof, window, and door textures), all individual textures should be packed into one image and sent as one file. A further simplification leading to a smaller file size is the use of a color palette instead of the true color quality. Thanks to color interpolation performed during the final texture mapping, many new colors emerge and the limited size of the original color palette is not recognizable.

5 Interactive Presentation on the Client Side

All data describing a virtual city can be stored in a distant database in arbitrary format. The sectors and the house data structures specified in previous sections were defined independently. Conversion to a specific format takes place when the data is sent from a server to a client and finally rendered. We use VRML terminology in the following text, although other technologies can also be applied. The reason why we prefer VRML is that it is an open, platform-independent, and well-known format. We intentionally avoid issues of copyright, which is not protected in the VRML specification, that is, data transferred to a client cache in the form of VRML files can be easily decoded and reused by anybody. Use of the VRML standard is welcomed by ordinary users, since they can install a VRML browser once and then browse many virtual worlds. The necessity to install special copyright-protected browsers for each city on the web is very inconvenient.

The following text assumes a standard VRML browser installed in a client computer. The additional "intelligence" needed for sector management and other tasks has to be programmed in the VRML Script node, using either ECMAScript (formerly JavaScript) or Java language. The general idea is that the user first downloads one starting sector,[1] and then the system loads other necessary data on the fly depending on the user's movement in the virtual environment. Another way is to request a server to send a sequence of sectors representing a path between two places of interest. This is useful for virtual guided tours, or as a support for a "How to get to . . . ?" function from the user interface.

The Script node performs two main tasks—it watches the avatar's activities, and manages the data contained in the sectors. Newly downloaded sectors are included into an existing scene graph, while unnecessary data is disposed of. Typically, a sector is loaded through the Inline node. If the sector has to be removed, we have to delete the corresponding Inline node (in the case of VRML 97) or to set the load field in the Inline node to FALSE (in the case of X3D). Actually, the memory management is not fully controlled by a Script programmer but internally by a VRML browser.

5.1 Dynamic Data Management

Since every sector contains a list of sectors (called Lvis in the previous text) that are either directly visible or somehow important, the strategy for generating requests to a server seems to be straightforward. One request is sent per each sector from a given Lvis list. This simple strategy fails for models with fine granularity of the sectors. When a sector represents a street connecting two squares, both squares are surely contained in the Lvis, but most of the other streets leading from those squares are not directly visible from the given sector. Thus, entering a square would lead to an incomplete view of the square area without the houses belonging to the other street sectors. Neither visibility information nor importance information from the Lvis list is able to cope with this problem. The solution is based on the topology, that is, the connectivity among the sectors. As the whole topology graph is usually not present in memory, necessary adjacency information can be indirectly retrieved from the visibility information.

Let VIS(C) be a set of sectors contained in the Lvis list of sector C. Let L_i^C denote a set of sectors that have to be loaded into a client computer memory for a given level i and sector C. A simplified notation L_i is used when C is unambiguously given. Then the sets L_0, L_1, L_2, . . . , L_n can be iteratively constructed as follows:

$L_0 = \{C\}$ A set containing only a given sector C

$L_1 = \bigcup_{C_j \in L_0} \mathrm{VIS}(C_j)$ The Lvis list for a given sector C

$L_2 = \bigcup_{C_j \in L_1} \mathrm{VIS}(C_j)$ The previous set extended by neighboring, visible, and other important sectors

. . . . . .

$L_n = \bigcup_{C_j \in L_{n-1}} \mathrm{VIS}(C_j)$ All sectors of a city (for sufficiently large n)

A high number i indicates that more sectors are loaded. Starting from the zero level representing the current sector only, further and further neighboring sectors are requested. This process of breadth searching can in theory continue up to loading the whole city, but in practice it can be stopped at level 2, which covers a satisfactorily large combination of directly visible sectors and neighbors of neighbors. The number i determining the L_i set can be dynamically changed according to the performance of the client computer. It is a more robust tool for controlling the rendering speed than the finer LOD technique. These two methods—sets of sectors (L_i) and LOD—can be efficiently combined to achieve smooth rendering. The cardinalities of L_i sets grow relatively slowly for historical towns with an irregular street structure, and rapidly for well-arranged modern cities. The upper limit ($i = n$) is less than or equal to the eccentricity of the current sector (node) in an adjacency graph.

Breadth search processes utilizing L_i sets can also be successfully employed if no visibility algorithm has been executed in the preprocessing phase. Lvis lists for all sectors should then contain close neighbors, that is, sectors whose IDs were assigned to gates.

1. Note that any sector can be set as the starting sector. There is no need to start a virtual walk always from the same point in the city.

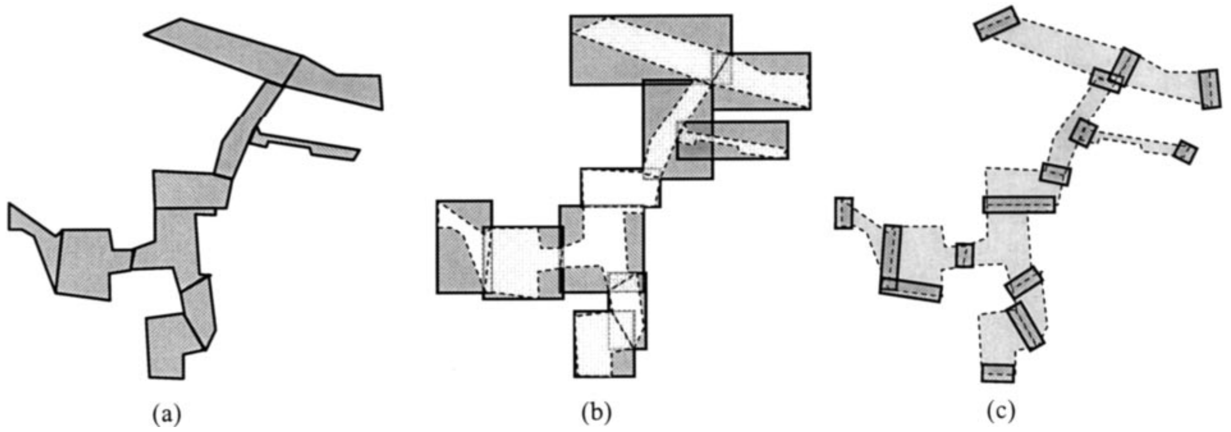


Figure 6. *User-in-sector tests based on (a) the point in polygon test, (b) the point in box test, (c) the point in gate test. Ten sample sectors have been taken from the map of Prague.*

5.2 Observing the Movement of an Avatar

Each time a user passes from sector C_{previous} to the neighboring sector C_{new}, the L_i associated with the newly visited sector has to be evaluated. Differences between previous and current L_i sets produce changes in the scene graph. Sectors belonging to the set $L_i^{\text{Cnew}} - L_i^{\text{Cprevious}}$ have to be downloaded while sectors from the set $L_i^{\text{Cprevious}} - L_i^{\text{Cnew}}$ are disposed of. Continuous observation of the position of the avatar in virtual space guarantees timely generated requests for new sectors and hence smooth presentation of the city.

Since the avatar-in-sector test is performed repeatedly, it should be optimized. Figure 6 shows three different methods.

5.2.1 Point in Polygon. Point in polygon is a test for a general polyhedron that has no support in VRML. Evaluation via an external program is necessary, thus making this test slow.

5.2.2 Point in Bounding Volume. The proximity sensor in VRML supports the test for point in bounding volume. A bounding box with arbitrary orientation can be used. The bounding boxes of neighboring sectors often overlap. Fortunately this is a positive rather than a negative feature, as will be explained below.

5.2.3 Point in Gate. Point in gate is actually a variation of the point in bounding volume test. From the mathematical point of view, the gate is a planar face, but in practice it is modeled by a box. While each bounding box used in the previous test belongs to exactly one sector, a box defined by a gate belongs to two sectors. For this reason, a simple point in box test is not sufficient to determine the direction of an avatar and a newly visited sector. Instead, we have to manage a sequence of box faces intersected by the avatar in order to evaluate the sector containing the avatar. Although there is no direct support in VRML for this special test, a gate box can be modeled using two half-gates represented by a pair of proximity sensors and additional script that evaluates the direction in which the avatar left the gate.

Note that a grid could be used as an auxiliary data structure for all tests. These tests can be further optimized in such a way that only the current sector and its neighbors are taken into consideration. Instead of *k* tests performed for *k* sectors loaded in memory, the number of tests drops to the number of gates of the current sector, typically from 2 to 4. This kind of optimization can be done under the condition that the avatar walks continuously. When rapid jumps to viewpoints in farther sectors are allowed, then all *k* tests have to be executed. Moreover, point in gate tests fail for such

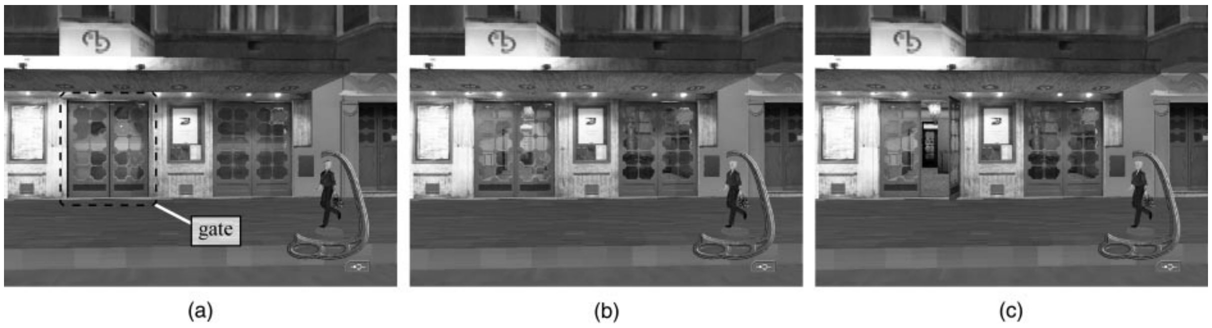


Figure 7. *Moving from the exterior to the interior in e-Agora application. (a) Avatar in a street sector. (b) The impostor behind the door is illuminated when the geometry of the next sector representing the entrance hall is being loaded. (c) A new geometry has replaced the impostor behind the door.*

jumps, since VRML browsers skip collision detection when switching to another viewpoint.

When testing users' behavior in a virtual environment we have to consider the following unfavorable situation. When a user enters a new sector, the system starts to evaluate the L_i set, updates the sectors in memory, and generates requests to a server. Due to the number of operations performed, the rendering speed drops significantly. Some users get the feeling that something is wrong in the direction that they are examining. They tend to take a step back, but this movement causes a return to the previous sector, and demanding computations start again. To overcome this particular problem, we can either delay updates of the scene graph or we can detect the avatar's position in a new sector not immediately on the border but somewhere inside. For simplicity, we have used the second approach. Both kinds of tests—(b) and (c) from Figure 6—can be easily modified. A straightforward solution is to slightly scale up the bounding boxes or gate boxes, respectively. Then the event that activates the dynamic loading of new data is generated at the time when the avatar leaves the box, either the current bounding box for test (b) or a gate box for test (c). In all cases, the avatar's position is relatively deep in the next sector, and repeatedly performed updates near borders are suppressed.

To conclude this section, let us demonstrate the use of gates in another web application. In the multi-user application called e-Agora (Adamec et al., 2001), users visit both exterior and interior scenes. The sectors describe either rooms in a virtual cultural center or a street segment around that building. Sectors are again connected via gates, which are represented by 3D models of doors in this case. Impostors are placed behind doors and are partially visible through semitransparent glass parts of the doors. When a user clicks on a door (via the TouchSensor node) a request to load the sector behind that door is generated. At the same time, the glass part of the door becomes more transparent and the impostor is illuminated by a light source. This is like switching on a light in the neighboring room as positive feedback to the user's action. The door is then slowly animated and is fully opened at the time when the neighboring sector has been loaded. The geometry of the new sector replaces the impostor and the avatar can pass through the door. When the avatar moves deeper into the sector, an event for closing the door is generated, all data of the previously visited sector is released from memory, and another impostor is placed on the opposite side of the door.

The whole process is depicted in Figure 7. It should be stressed that this particular web application utilizes a framework of sectors and gates, but all the vast geometrical and texture data has to be installed locally in advance. The web is primarily used for social interaction (chatting, meetings with individual 3D avatars, etc.), but not for transferring the 3D scene data.

6 Open Problems

Almost every component used in the processing pipeline depicted in Figure 1 contains areas that need to be improved. The most demanding requirements are still placed on research in Computer Vision and Computer Graphics. In the 3D reconstruction area, we need to solve the following issues:

1. Algorithms that generate meshes adapted to real shapes of buildings. Such meshes should contain large planar faces on walls and roofs, but dense triangles for architectural details.
2. Reconstruction based on knowledge of the house structure. Repeatedly used elements like windows should be recognized and optimized for further LOD creation. Fully automatic creation of LODs for houses is a highly important task.
3. Since real cities do not consist of houses only, we need to automatically recognize other objects such as street lamps, trash cans, trees, park benches, and so forth. Once recognized, these objects can be removed from photos that are used for reconstructions of buildings, but they have to be added to a 3D model of the city and presented to visitors. From a few 3D model prototypes we can efficiently generate a number of duplicates in the scene. On the other hand, many visual obstacles such as cars, dogs, and people can be removed forever.

In web 3D visualization, the following tasks are of great interest:

1. Automatic determination of sectors based on a city map (GIS database) would save a lot of time when creating a virtual city database. Several constraints influence the definition of sectors. A data package containing all objects attached to the sector has to be of limited size (web constraint). The next constraint concerns the shape of the sector and the number of gates (visibility issues).
2. A seamless combination of impostors and 3D models is required for open places like hills, observation towers, islands in rivers, and so forth. An efficient mixture of image based and model based rendering algorithms should be implemented in the client browser/viewer.
3. A flight over a city requires a special solution. Aerial photos mapped on a terrain can be combined with highly simplified house models. The use of LOD is an obvious condition. The question is how to transfer an extremely high number of houses from a server in the shortest time. Instead of many individual files (one per house), the models need to be grouped before they are sent through the net.

7 Conclusion

Various issues concerning preparation and presentation of 3D city models on the web have been discussed in this paper. Web specific limitations require the adaptation of already existing algorithms and the use of appropriate data structures. We have introduced a framework for virtual city models taking into account historical towns with an irregular street structure. The framework is made of a set of city sectors holding models of houses and other spatial objects. A special urban LOD has been designed to achieve both fast download of house models and real-time rendering on the client side. Selected methods for dynamic management of sectors and for observing the movement of a user were presented.

In this study we have presented our experience from the design, implementation, and operation of the Virtual Old Prague web application. This project prefers to use standard, open technologies rather than proprietary and perhaps more efficient solutions. These standard technologies are VRML, HTML, JavaScript, and Java on the client side; PHP and mySQL database on the server side. The area of Prague that is modeled as a virtual city consists of about 25 streets, 6 larger squares, 350 houses, and tens of additional 3D objects arranged into 80 sectors. Although these numbers are not astonishing, the framework is scaleable and can grow up to an entire city. Currently, two European cities—Graz in Austria, and Bratislava in Slovakia—are testing the Virtual Old Prague project with the aim of using this technology for presenting the historical kernels of the cities on the web.

Acknowledgments

This work has been partly supported by the Ministry of Education, Youth, and Sports of the Czech Republic under research program No. 6840770014 (Research in the Area of the Prospective Information and Navigation Technologies).

References

Adamec, J., Cizek, J., Masa, M., Silondi, P., Smetana, P., & Zara, J. (2001). Virtual house of European culture: e-AGORA. *Virtual Storytelling*. Berlin: Springer, 208–211.

Bittner, J., Wonka, P., & Wimmer, M. (2001). Visibility preprocessing in urban scenes using line space subdivision. *Proceedings of Pacific Graphics (PG'01),* 276–284.

Collins, R. T., Jaynes, C. O., Cheng, Y. Q., Wang, X. G., Stolle, F. R., Riseman, E. M., et al. (1998). The ascender system: Automated site modeling from multiple aerial images. *CVIU, 72*(2), 143–162.

Coorg, S., & Teller, S. (1999). Extracting textured vertical facades from controlled close-range imagery. *Proceedings of CVPR,* 625–632.

Decoret, X., Schaufler, G., Sillion, F., & Dorsey, J. (1999). Multi-layered impostors for accelerated rendering. *Computer Graphics Forum, 18*(3), 61–73.

Durand, F. (2000). A multidisciplinary survey of visibility. *Visibility, problems, techniques, and applications.* ACM SIGGRAPH Course notes.

Hoppe, H. (1996). Progressive meshes. *Computer Graphics (ACM SIGGRAPH) Annual Conference Series,* 99–108.

Koch, R., Pollefeys, M., & Van Gool, L. (2000). Realistic surface reconstruction of 3D scenes from uncalibrated image sequences. *Journal Visualization and Computer Animation, 11,* 115–127.

Kunii, Y., & Chikatsu, H. (2003). Building extraction and modeling in urban area by image sequence analysis. *Proceedings of SPIE Electronic Imaging 2003—Videometrics VII, 5013,* 186–193.

Luebke, D., & Georges, C. (1995). Portals and mirrors: Simple, fast evaluation of potentially visible sets. *Proceedings of Symposium on Interactive 3D Graphics '95,* 105–106.

Lynch, K. (1960). *The image of the city.* Cambridge, MA: MIT Press.

Moons, T., Frere, D., Vandekerckhove, J., & Van Gool, L. (1998). Automatic modelling and 3D reconstruction of urban house roofs from high resolution aerial imagery. *Proceedings of ECCV 98,* 410–425.

Sillion, F., Drettakis, G., & Bodelet, B. (1997). Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum—Eurographics '97, 16*(3), 207–218.

Slater, M., Steed, A., & Chrysanthou, Y. (2002). *Computer graphics and virtual environments: From realism to real-time.* Reading, MA: Addison Wesley.

Taylor, C. J., Debevec, P. E., & Malik, J. (1996). Reconstructing polyhedral models of architectural scenes from photographs. *Proceedings of ECCV 96,* 659–668.

Zara, J., Chromy, P., Cizek, J., Ghais, K., Holub, M., Mikes, S., et al. (2001). A scaleable approach to visualization of large virtual cities. *Proceedings of the Fifth International Conference on Information Visualisation (IV 2001),* 639–644.

Zheng, J. Y., & Tsuji, S. (1998). Generating dynamic projection images for scene representation and understanding. *Computer Vision and Image Understanding, 72*(3), 237–256.

Zheng, J. Y., & Shi, M. (2003). Mapping cityscapes to cyber space. *Proceedings of the 2003 International Conference on Cyberworlds,* 166–173.

Zlatanova, S., & van den Heuvel, F. A. (2002). Knowledge-based automatic 3D line extraction from close range images. *International Archives of Photogrammetry and Remote Sensing (ISPRS), 34*(5), 233–238.