Real-Time Visualisation of Urban Landscapes Using Open-Source Software

Martin Kada[1], Stefan Roettger[2], Karsten Weiss[2], Thomas Ertl[2], Dieter Fritsch[1]

ifp[1] / VIS Group[2], University of Stuttgart, Germany
martin.kada@ifp.uni-stuttgart.de roettger@cs.fau.de

Abstract

The paper presents the results of the project GISMO, which aimed on generating and interactively visualising a 3D urban landscape model of the city of Stuttgart, Germany. With respect to the desired flexibility to support walkthrough and flyover applications, a combined approach using continuous level of detail, the impostor technique and a method for generalizing 3D building models was used to speed up the visualization. To reduce the costs of the project, the data collection tools and the visualization environment was built solely with open-source software.
Keywords: Urban Landscapes, OpenSceneGraph, Impostors, Terrain Rendering, 3D Generalization.

1 Introduction and Related Work

The advances in automatic data acquisition of urban sceneries have lead to an increasing amount of data covering large areas. Better and larger models are important for application areas like urban planning, emergency response, tourism, entertainment, traffic management, construction of large-scale projects, and education. In these areas the interactive visualization of the urban models is of great importance for an in-depth analysis of the data set.
In the recent past, mainly three methods to manage real-time visualization of entire city models have been described. First the more traditional systems reduce geometric complexity by appropriate LOD selection and management [13]. Secondly, there exist a variety of image based algorithms which improve rendering performance. Here the impostor [6, 7, 8] based approaches [9] are very popular due to their simplicity and high performance. And finally, efficient occlusion culling algorithms [11, 12] have been devised for the special application area of urban landscapes.

2 Motivation

In public institutions the need for high-performance visualization systems at low cost has been growing in the last years. Many of the larger cities (such as Graz, Vienna, Saarbrücken and Stuttgart to name only a few in Central Europe) have been collecting digital urban data, but do not have efficient visualization environments that could ease urban planning for example.
In our specific case, the city of Stuttgart has acquired a large city model and now has the demand for visualizing the city model at low cost. Optimally, the visualization software should run on every up-to-date PC equipped with sufficient memory and a commodity graphics accelerator.
In order to evaluate the applicability of our aims, the project GISMO was set up for the ”Real-Time Visualization of 3D Urban Landscapes”. The project was carried out as a cooperation of the VIS Group and the ifp at the University of Stuttgart. For the reduction of costs we planned to use open-source software and commodity graphics hardware. With respect to the desired flexibility the support for both walkthrough and flyover applications was an additional goal. Since occlusion algorithms are efficient for walkthroughs but do not yield high performance gains for flyovers, we reckoned that the use of impostors would be the most promising technique for our project as outlined in Section 6. We also choose to gain performance by using geometrical simplification algorithms for the buildings, but since our buildings consist of only one to a few dozen of triangles on the average, it is difficult to gain significantly more performance without altering the buildings' appearance (see Section 8 for more information on this topic).

3 Data Acquisition

The data set that we use contains the visual representation of the city of Stuttgart and the surrounding area of the size 50×50 km. It includes a 3D city model provided to us by the City Surveying Office of Stuttgart, digital terrain models at a resolution of 10 meter for the extent of the inner city and 30 meter for the surroundings. The corresponding aerial and satellite images have a ground pixel resolution of 0.8 and 5 meter, respectively. The covered area is chosen so that the visualization stretches as far as the virtual horizon.
The 3D city model of Stuttgart was photogrammetrically reconstructed in a semi-automatic process. For data capturing, the building ground plans from the public Automated Real Estate Map (ALK) and the 3D shapes measured from aerial images were used [10]. The resulting wireframe model contains the geometry of 36,000 buildings covering an area of 25 km^2 meaning that almost every building of the city and its suburbs is included (also compare Figure 1). The overall complexity of the model amounts to 1.5 million triangles. In addition to the majority of relatively simple building models, some prominent buildings like the historical New Palace of Stuttgart are represented by 3,000 (and more) triangles.
To improve the visual appearance, we captured the facade textures of 500 buildings that are located in the main pedestrian area. Approximately 5,000 ground based close-up photographs of the building facades were taken using a standard digital camera. The textures were extracted from the images, perspectively corrected, rectified, and manually mapped to the

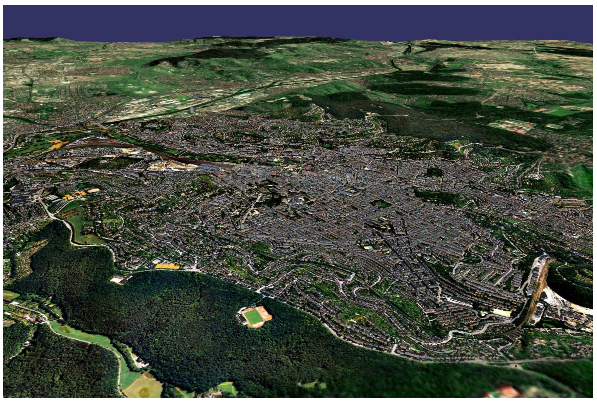

Figure 1: Overview of the Stuttgart city model. A total of 36,000 building models are available for the densely populated area bordered by the woods at the top and bottom of the image.

corresponding planar facade segments. To simplify the task of texture placement, we decided to extend our visualization environment with a specialized component. The user selects the facade and the corresponding texture, which is then initially snapped to the bounding box of the geometry. A user controlled affine transformation precisely adjusts the final texture coordinates. Using this approach, we managed to process the aforementioned 500 buildings in roughly 20 man-months. Because of the large size of the original texture dataset, we down-sampled the textures to a resolution of approximately 15 centimeters per pixel. Buildings with no real captured facade textures were finally colored randomly with different colors for the facade and the roof.

4 Visualization Techniques

Due to the large amount of geometry and texture data, a brute force rendering approach is not suited even for current high performance 3D graphics accelerators. It is therefore inevitable to use acceleration techniques like visibility culling, level-of-detail (LOD) representations and image based rendering in order to speed up the visualization process.

Occlusion culling is a very popular technique for urban walkthroughs. It is especially efficient in situations where large areas are occluded by close-by buildings. Since this is not our main application scenario, this technique was, however, not realized. Instead we decided to use an image-based rendering approach for the visualization of the building objects. By the use of impostors, both the geometry and the amount of the highly detailed facade textures that are visible in the scene are reduced during rendering.

5 Open Scene Graph

The Open Scene Graph (OSG), which recently has become a powerful alternative to traditional tools like Performer, integrates the impostor concept as a specialized level-of-detail node which is usable right out of the box. The Open Scene Graph is a cross-platform C++ / OpenGL library for real-time visualization. It is developed and maintained mainly by Robert Osfield and is freely available under the GNU LGPL at [3]. We use OSG because the library not only features high-performance rendering capabilities and excellent support for PC graphics accelerators, but also offers a variety of utility classes like GUI support, camera manipulators, picking functionality and loaders for many common data formats.

The most important reason for choosing OSG however was its clean design, the extensibility and also the OSG source code which was available right from the start of our project. Having the code on-hand we were able to develop a first impostor implementation for OSG based on the original paper of G. Schaufler [6]. Our prototype performed well enough so that the impostors were later integrated into the official OSG release.

6 Impostors

Impostors are an image-based rendering technique. Like a billboard, an impostor replaces a complex object by an image that is projected on a transparent quadrilateral. A common example for the use of billboards is the visualization of trees. Provided that the viewer stays close to the ground level, the image of a tree is a good approximation of the real geometry for all view points. As the viewer moves around the scene, the quadrilateral is rotated so that the image always faces forward. Because billboard images are created a priori and are therefore static, this technique can only be used for objects that look similar under rotation. In contrast to that, the images of impostors are dynamically generated by rendering the objects themselves for the current point of view. If consecutive viewpoints are close together, the impostor images of slowly moving objects that are located far from the viewer do not change notably with every frame. From this it follows that those impostor images can be reused for several frames and therefore speed up the overall rendering process. In OSG the impostor technique is implemented as a discrete LOD node class. Depending on a user-defined distance threshold, the object is either rendered traditionally or as an impostor image. The recomputation of the impostor image is performed automatically by OSG using a very similar error ciriterion as the one proposed by Schaufler. Texture management is also done automatically by OSG, so that the user basically only has to add an impostor node above the appropriate objects in the scene graph.

A 3D city model consists of vast amounts of building objects. In an urban scene, these innately static objects extend over a large area so that only a fraction of the objects is actually close to the viewer. It can be assumed that in this context the majority of the building objects is located far enough from the viewer to cause few image updates and can therefore be visualized efficiently by impostors. The use of impostors results in a texture memory overhead, however, because the impostor images additionally occupy valuable texture space on the graphics hardware. To limit the additional memory usage, impostors must consequently not replace single building objects, but rather several buildings that are located close together.

We used a very simple approach to arrange the building data in our scene graph. The test area is divided into a regular 2D grid and building objects whose centroids are located in the same cells are grouped together. We also did some testing with hierarchically organized impostors, but did not find them to be superior in our context. The additional hierarchy levels noticeably reduced the image quality due to multiple filtering and the texture memory overhead increased even further.

7 Terrain Rendering

For the visualization of the digital terrain model on which the buildings are placed, a continuous level-of-detail (C-LOD [2]) approach is used. We developed a terrain rendering library using C-LOD in a previous project, which was named libMini. In the meantime the library has become open source and integrates easily with OSG (by calling the terrain render right before the main OSG render action).

The library is licensed under the terms of the GNU LGPL and can be downloaded from the home page of the author [5]. The library implements the C-LOD approach as described in [4]. It also allows suppressing the popping effect which is due to the view-dependent simplification of the C-LOD scheme. The suppression of the popping effect is achieved by a technique called geomorphing. This is important since the rendering of the buildings consumes most of the time slice and there is little time left to render the terrain. As a consequence, the error threshold of the view-dependent simplification has to be set to high values to produce as few triangles as possible. Without geomorphing the rather coarse triangulation would lead to excessive popping.

The main advantage of the use of the mentioned terrain rendering library is that the original terrain data, that is high resolution height fields, can be used without a conversion to triangle meshes. Due to the availability of terrain data for a large area (2,500 km^2) the viewer is able to see the real horizon beyond the city model.

Because of the ultra high resolution of the original data we organized the terrain as a 12 by 12 regular grid. Each tile of the grid has a resolution of 10, 20, 40, or 80 meter depending on the distance to the city center. The texture resolution is approximately 1 meter for the city center, and is decreasing gradually to 2, 5, and 10 meters in the perimeter. The C-LOD library supports tiled terrains with different grid resolutions in an efficient way and takes care of cracks that might appear at the tile borders. The corresponding textures of each tile were down-sampled in a similar fashion to a maximal resolution of approximately 1 meter. With S3TC texture compression the textures accounted for approximately 52 MB of graphics memory (208 MB uncompressed).

8 Generalization

As noted previously the simplification of the 3D building models is difficult to yield better rendering performance. Since the vast majority of the buildings consist of only a few quadrilaterals, the buildings cannot be simplified any further without altering the appearance noticeably. However, some of the most prominent buildings in the city center of Stuttgart consist of a substantial number of triangles. For those buildings we have devised an automatic generalization process [1] which eliminates data acquisition artifacts, groups elements that lie in nearly the same plane, and throws away small protrusions while keeping the building symmetries (see paper for more details). As an example, we reduced the 2930 triangles for the historical New Palace to 1837 triangles after generalization (see also Figure 2).

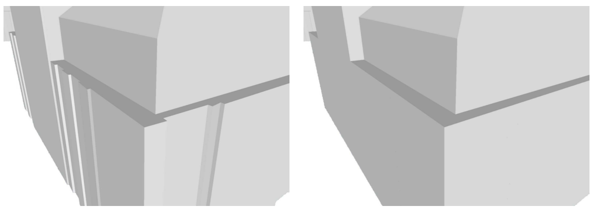
Figure 2: Before and after generalization of a 3D building model. Removed protrusions of the facade of the historical New Palace of Stuttgart.

9 Results

The experimental results have been measured on a standard PC equipped with a 2.0 GHz Intel Pentium P4 processor, 512 MB of memory and an NVIDIA GeForce4 Ti 4200 graphics accelerator with 64 MB of graphics memory.
Without impostors (frustum culling alone) we achieved 2 fps brute force rendering performance for the entire data set including terrain rendering. Enabling OSG impostors led to a speed up that depends on the size of the regular impostor grid, and thus the number of used impostors. In principle, the performance increases with the number of used impostors, but one should not use too many impostors, since OSG has a computational overhead for each. One also has to keep in mind that each impostor consumes additional texture memory, which may have to be paged in and out of the dedicated graphics memory. On the other hand, decreasing the number of impostors by increasing the tile size of the regular impostor grid has the effects that too many objects are grouped into a single large impostor. In such a setup the recomputation of the impostors is triggered too frequently, so that the performance drops significantly.
The best results were achieved with a regular layout of 60×60 impostors (see Table 3). Enabling OSG impostors with a distance threshold of 1 km boosted the performance to 11 fps on the average, but the recomputation load for the impostors was still quite high in some cases which led to occasional frame drops. Increasing the impostor distance threshold to 1.5 km removed most of the frame drops, but since fewer buildings were replaced by impostors the frame rate dropped to approximately 9 fps. With a distance threshold of 2 km the occasional frame drops were eliminated completely, but the frame rate was only 7 fps. But even in the last case we experienced a speed up of 350% which is quite notable for the seemingly simple strategy.

grid layout	impostor distance threshold	frame rate
1×1	brute force	2 fps
60×60	brute force	2-3 fps
60×60	1,000m	11 fps
60×60	1,500m	9 fps
60×60	2,000m	7 fps

Figure 3: Rendering performance with and without impostors.

Figures 4 and 5 show screen shots of the city center of Stuttgart rendered with the described impostor technique. The visibility of an impostor update is very low since the update happens quite far away (> 1 km). In many cases the updated impostors are occluded by non-impostor buildings, so that the update is

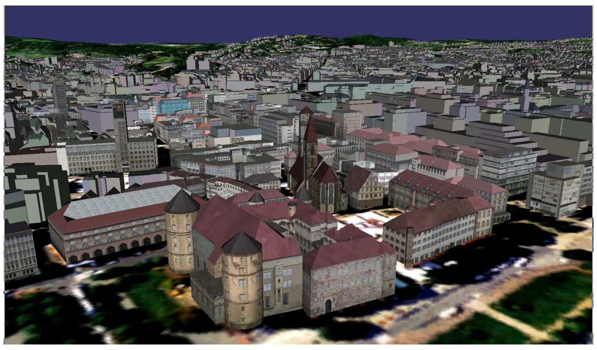

Figure 4: View southwards from the Old Castle of Stuttgart.

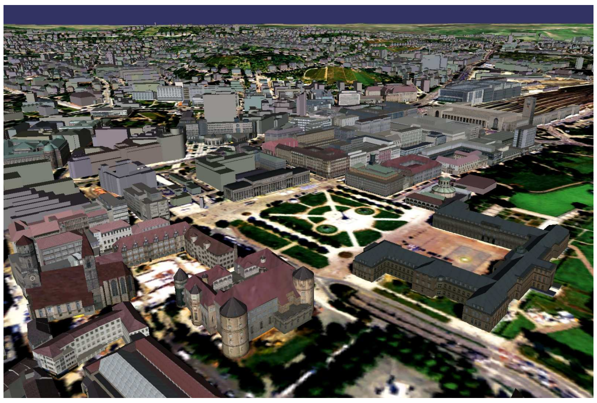

Figure 5: View from the city center toward the virtual horizon.

hardly noticeable. For the same reasons the z-fighting of the impostors with the underlying terrain is hardly visible as well.

10 Conclusion and Future Work

The GISMO project has demonstrated that real-time visualization of large urban scenes is possible using merely open source software. The libMini and OSG libraries are both capable of rendering huge data sets at interactive frame rates and are thus good alternatives for commercial products. In combination with commodity PC hardware, cheap but powerful visualization systems can be built. Due to the increasing availability of spatial 3D data, a detailed urban landscape model was realized in GISMO that covers a very large area. The user can experience the virtual scene that stretches as far as the virtual horizon both in flyover and walkthrough mode. In order to accomplish this task, we combined the C-LOD terrain rendering approach with the impostor technique to improve the rendering performance.

In future work we strive to optimize the impostor implementation of OSG to circumvent the mentioned problems. We also plan to include vegetation and dynamic objects (cars, pedestrians). Another goal is to apply the automatic generalization of the building models to the entire data set and not only a few prominent buildings.

11 Acknowledgements

We thank Robert Osfield for his help on our first implementation of the impostors and their final integration into OSG. We acknowledge Mathias Jahnke, Katharina Schmid, Laurentin Blaga, Emese Farkas, Cosmin Dragomir and Philipp Oswald who captured the numerous facade textures that are used in our 3D city model.

References

[1] Martin Kada. Automatic Generalisation of 3D Building Models. *Joint International Symposium on Geospatial Theory, Processing and Applications '02*, 2002.

[2] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. In *Proc. SIGGRAPH '96*, pages 109–118. ACM, 1996.

[3] Robert Osfield. OpenSceneGraph. *www.openscenegraph.org*, 2003.

[4] S. Roettger, W. Heidrich, Ph. Slusallek, and H.-P. Seidel. Real-Time Generation of Continuous Levels of Detail for Height Fields. In *Proc. WSCG '98*, pages 315–322. EG/IFIP, 1998.

[5] Stefan Roettger. libMini Terrain Rendering Library. *wwwvis.informatik.uni-stuttgart.de/˜roettger*, 2003.

[6] G. Schaufler. Dynamically Generated Impostors. In *Proc. GI Workshop on Modeling, Virtual Worlds, and Distributed Graphics '95*, pages 129–136, 1995.

[7] G. Schaufler. Per-Object Image Warping with Layered Impostors. In *Proc. 9th Workshop on Rendering '98*, pages 145–156. Eurographics, 1998.

[8] J. Shade, S. Gortler, L. He, and R. Szeliski. Layered Depth Images. In *Proc. SIGGRAPH '98*, pages 231–242. ACM, 1998.

[9] F. Sillion, G. Drettakis, and B. Bodelet. Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery. *Computer Graphics Forum (Proc. of Eurographics '97)*, 16(3):207–218, 1997.

[10] Manfred Wolf. Photogrammetric Data Capture and Calculation for 3D City Models. *Photogrammetric Week '99*, pages 305–312, 1999.

[11] Peter Wonka and Dieter Schmalstieg. Occluder Shadows for Fast Walkthroughs of Urban Environments. In *Proc. Eurographics '99*, pages 51–60, 1999.

[12] Wonka, P. and Wimmer, M. and Sillion, F. Instant Visibility. In *Proc. Eurographics '01*, pages 411–421, 2001.

[13] Christopher Zach. Integration of Geomorphing into Level of Detail Management for Realtime Rendering. *Technical Report, VRVis Research Center, University of Graz*, 2002.