City Virtualization

Gregor Fabritius, Jan Kraßnigg, Lars Krecklau, Christopher Manthei,
Alexander Hornung, Martin Habbecke, and Leif Kobbelt

Computer Graphics Group
RWTH Aachen University
Ahornstrasse 55
D-52074 Aachen
http://www.graphics.rwth-aachen.de

Abstract: Virtual city models become more and more important in applications like virtual city guides, geographic information systems or large scale visualizations, and also play an important role during the design of wireless networks and the simulation of noise distribution or environmental phenomena. However, generating city models of sufficient quality with respect to different target applications is still an extremely challenging, time consuming and costly process. To improve this situation, we present a novel system for the rapid and easy creation of 3D city models from 2D map data and terrain information, which is available for many cities in digital form. Our system allows to continuously vary the resulting level of correctness, ranging from models with high-quality geometry and plausible appearance which are generated almost completely automatic to models with correctly textured facades and highly detailed representations of important, well known buildings which can be generated with reasonable additional effort. While our main target application is the high-quality, real-time visualization of complex, detailed city models, the models generated with our approach have successfully been used for radio wave simulations as well. To demonstrate the validity of our approach, we show an exemplary reconstruction of the city of Aachen.

Keywords: Virtual City Models

1 Introduction

The efficient generation and visualization of complex virtual city models plays an increasingly important role in a variety of computer graphics applications including virtual city guides and geographic information systems [Gis08], computer games, or large scale visualization tools like Google Earth [Goo08]. However, the creation of these models, e.g., using 3D modeling tools [Ble08] is a difficult problem which requires considerable manual effort and expert knowledge.

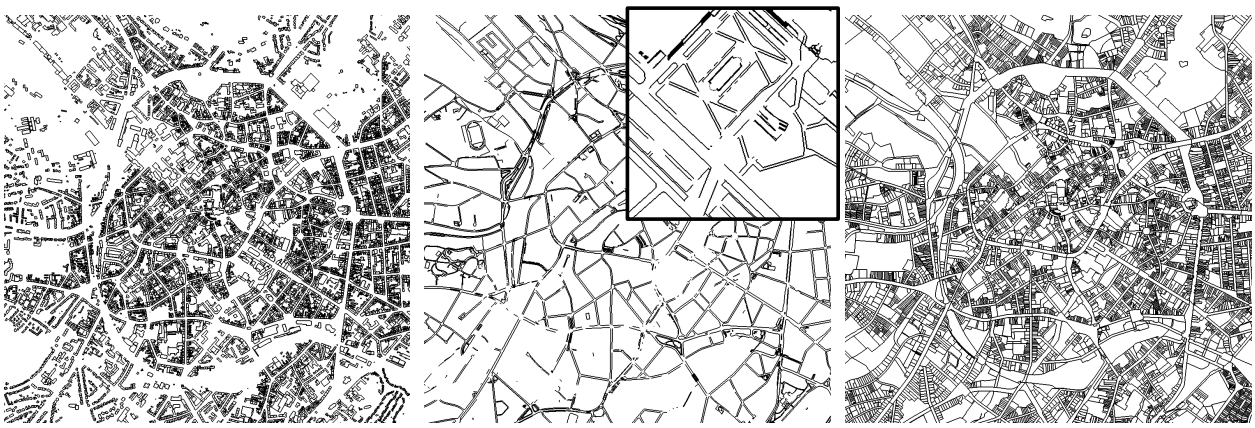
Recently, there have been a number of significant advances in computer graphics and vision research in order to automate the generation of complex city models. On the one hand we find techniques for procedural [PM01, MWH+06] and grammar based modeling [WWSR03]

of cities. Similar systems have been proposed for generating pseudo infinite cities in real-time [GPSL03] or for automatic synthesis of facade textures [LDG01, LG06, MZWG07]. The fundamental basis for many of these techniques are shape grammars and patterns in architecture [Sti75, CAS77]. Although these methods are able to produce complex and detailed city models with considerable quality [WMV+08], they are often based on simplified models so that it is difficult to reconstruct the visual variety and structural patterns observable in real cities. Even more importantly, many applications such as virtual city guides or navigation systems require the integration of real-world data including, for example, correct street maps [Geo08, Ope08] or reconstructions of specific landmark buildings. Most automatic systems for procedural city generation do not address these important requirements and it is often difficult to consolidate existing automatic approaches with techniques for creating virtual city models from real input data.

On the other hand there are a variety of attempts to digitize 3D geometry and texture of real cities by employing computer vision techniques. A large body of work on city modeling is based on aerial images [SV01, VD01]. Although these techniques can capture the global city shape as well as the geometry of buildings, it is generally not possible to extract detailed and high quality textures for visualization. With recent advances in mobile tracking and efficient solutions for laser scanning and passive stereo reconstruction, it is now possible to build small scale acquisition systems. For instance, in [FJZ05, CLCG06, PNF+08] the authors mount reconstruction hardware onto a vehicle and reconstruct detailed street maps and textured facades. A semi-automatic approach for detailed architectural models from images has been presented in [DTM96]. These image-based approaches allow to model cities realistically at various levels of detail by employing actual photos of real buildings. Unfortunately these methods often focus on a particular subproblem and there is currently no single solution for image-based city reconstruction which properly addresses all involved issues. Moreover, image-based reconstruction techniques are not yet mature enough to support fully automatic data acquisition and processing, so that a considerable amount of manual work is still required in order to integrate data from different sources.

Hence, for creating virtual models of real cities, a certain trade-off between the correctness and the plausibility of a model has to be considered in practice: rule-based systems provide the tools for automatically generating the desired complexity, and image-based modeling enables a more realistic reproduction of buildings. Moreover, there is an increasing pool [Geo08, Ope08] of data resources of street and building maps publicly available. Given these different data sources and approaches the main contribution of this paper is to identify and describe the essential steps for a flexible and scalable modeling pipeline to generate virtual city models. Our conceptual approach to city reconstruction and visualization consists of the following main steps:

Model Generation: The geometry of the main city model is generated using a hybrid approach consisting of 2D map data, a terrain map, and automatic rule-based systems for creating a 3D model from the 2D data.



(*a*) (*b*) (*c*)

Figure 1: Our input data consists of 2D maps of buildings (a), streets (b) and properties (c). The close-up of the streets shows typical inconsistencies such as holes.

Texture Processing: A framework for generating and processing large texture and material databases is used to create authentic textures for prominent city regions. The remaining areas are textured using a rule-based system as well.

Image-based Modeling: Complex landmark buildings which cannot be reconstructed from the 2D maps are modeled using image-based techniques.

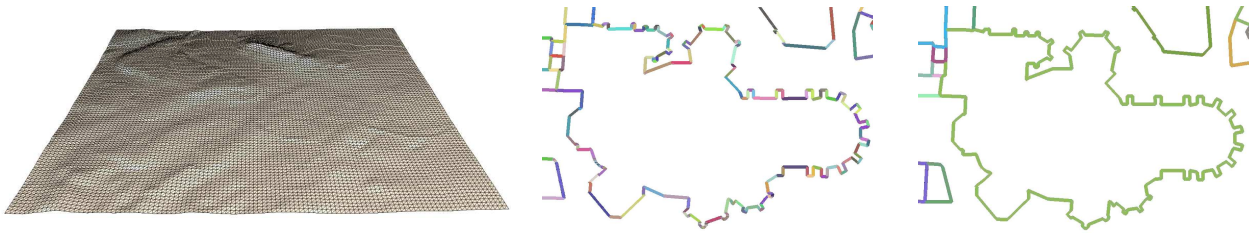
Visualization: A highly efficient approach based on deferred shading is used to render the resulting complex city models with global illumination effects in real-time.

The following sections describe these parts of our prototype in detail and present an exemplary reconstruction of the city of Aachen.

2 Model Generation

The automatic model generation is based on 2D street and building maps of Aachen which were in our case provided by the land registration office in form of multi-layered vector maps [Geo08]. The provided data contained separate layers with 2D line segments representing streets, buildings, and properties (Fig. 1). In addition to that we had a height map of the region of Aachen (Fig. 2 a).

Unfortunately the provided 2D data contained, in our case, a considerable amount of degeneracies such as isolated line segments, holes, redundant vertices, or undesirable intersections between lines. For instance, the building map does not contain consistent information about individual buildings, but consisted of a set of 1.2M unconnected edges. Hence, the first step of the model generation requires a pre-process in order to repair the 2D data. We employ basic graph algorithms for cycle detection to identify line loops which are part of a single building (Fig. 2 b,c). Similar problems occur with the road map: street segments ended before crossings, contained holes, or other types of artifacts such as undesirable line crossings.



(a) (b) (c)

Figure 2: Height map of the region of Aachen (a). Unconnected input edges before (b) and after loop detection (c).

We address this problem geometrically by extending disconnected line segments and computing proper line intersections. Using these techniques, most of the defective data can be repaired automatically. However, for a small number of remaining ambiguous or inconsistent configurations it turned out that simple manual intervention for joining street segments is more effective than trying to address all inconsistencies automatically.

Given the repaired data, the next major step is the extension of the 2D polygons representing buildings to actual 3D geometry. Since our base data does not provide any information about the building heights, we implemented a simple rule-based system which assigns varying numbers of floors to each building. The height computation is additionally influenced by the base area of each building, based on the assumption that the area and height of a building are correlated. Finally we adjust the height of neighboring buildings to improve the visual appearance. Given the building height, the actual 3D building is created by duplication and extrusion of the 2D base polygon. An additional simple roof geometry is added to rectangular buildings. Here again, neighboring houses are considered in order to align the roofs within a single street.

The last processing step consists of an orthogonal projection of the 3D building models and the 2D streets and properties onto the terrain data in order to produce the final mesh. To retain the edges of these input polygons in the output tesselation we compute a constrained Delaunay triangulation of the terrain and the 2D vector data. This is essential in order to differentiate between the different types of terrain such as streets or green areas during visualization. Using this approach we create two different types of output models: a watertight triangle mesh (Fig. 3) including the complete geometry for mobile network simulations, and two separate meshes for the buildings and the terrain optimized for culling during the visualization.

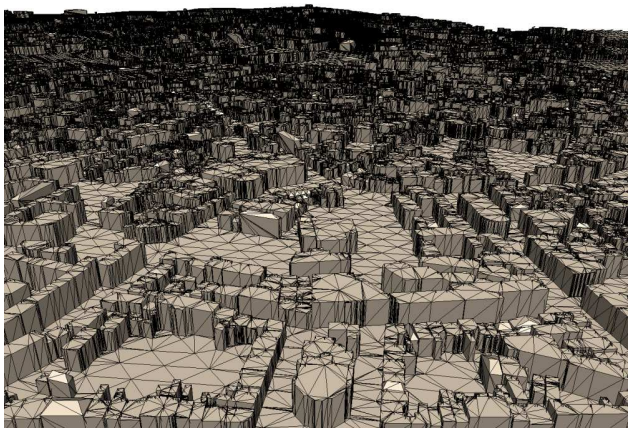


Figure 3: The resulting 3D mesh.

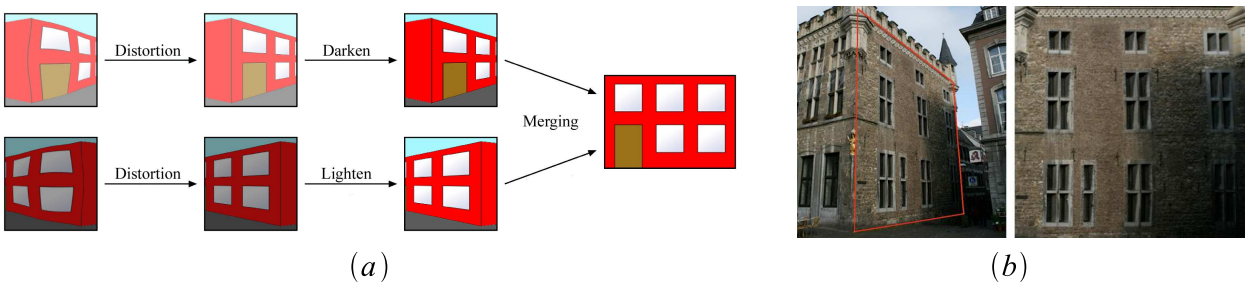


Figure 4: The workflow for texture processing (a). Example for image rectification (b).

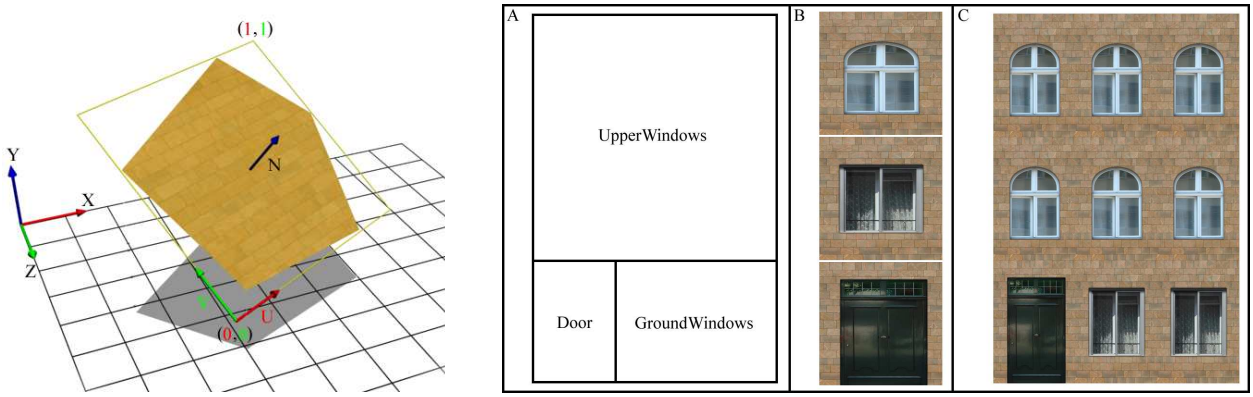
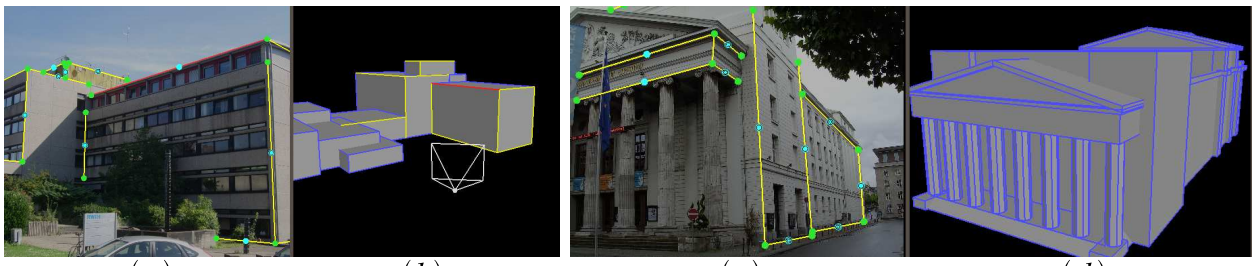


Figure 5: Left: computation of texture coordinates for walls and roofs. Right: combination scheme (a) for tile elements (b) to build complex facades (c).

3 Texture Processing

Real world photos are generally subject to a number of undesirable effects such as illumination changes or image distortion. These effects prevent a straight forward generation of high quality textures. We developed an image processing pipeline based on a set of image processing filters or modules which address these issues. The input is a library of images taken from within the city which we acquired with a standard digital camera. Besides standard color filters for adjusting brightness, contrast and gamma of an image, our two most important filters address the pincushion distortion caused by the camera lenses and perspective effects [HZ04] in cases where it is not possible to take pictures with an orthogonal view onto the respective facade, e.g., due to occlusions (Fig. 4). Some processing modules might use more than one input image, so that we actually employ an acyclic processing graph instead of a simple linear pipeline. The module for merging images, for example, takes two input images in which the user can define a planar region that covers the same part of a facade. The system will merge those images automatically by rectification and blending.

After this pre-process, the raw images contained in the database are augmented with additional material parameters in order to simulate different surface types. These attributes include, for instance, color settings for ambient lighting, and different maps for the diffuse color or the specular intensity. Texture coordinates (U, V) of a polygon are computed automatically using the direction of greatest ascent in Y direction as the V axis in parameter



(a) (b) (c) (d)

Figure 6: The image based tool for modeling landmark buildings. Input image of the computer science department (a). 3D model and estimated camera position (b). Input view of the theater (c) and the generated 3D model (d).

space. The orthogonal U axis is then computed from the normal N and V (Fig. 5, left).

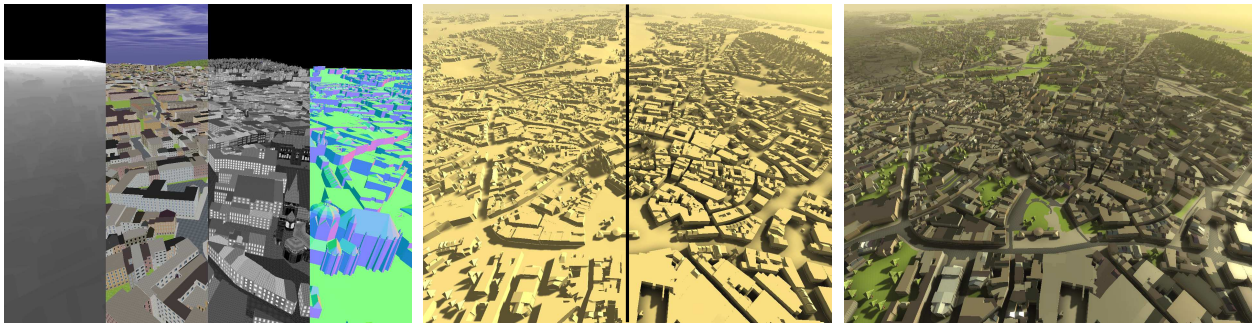
The mapping of materials from this database to the 3D model is done in two different ways: For prominent buildings or areas such as the market place in Aachen we assign materials and the corresponding textures manually. The remaining buildings are textured using an automatic procedural approach similar to [WWSR03] and [MWH+06] using tile elements. We distinguish four kinds of tiles: doors, ground windows, upper windows and walls. The procedural system splits the facades (Fig. 5, right) and assigns texture elements from the material database. In order to maintain the polygon count we do not explicitly split the geometry for window tiles but assign texture coordinates greater than one in order to create repeating texture patterns. The number of tiles is calculated by the real size of the seamless texture defined by the attributes of its corresponding material. Remaining small patches are simply textured using wall backgrounds.

Our material editor provides two layers of randomization for the procedural texturing. First the system selects a random library that covers the same architectural style so that the different walls of a single house get a similar look. Then the algorithm chooses a particular material in each category in order to make all tiles of the same kind look equal for the whole building. The probabilities for each library and material can be adjusted in order to control the stochastic distribution and variety of the different styles.

4 Image-based Modeling

Although the automatic model generation and the procedural texturing result in quite plausible city models, it is obviously not possible to use these approaches to model more complex landmark buildings such as the city hall or cathedral of Aachen which cannot be extruded from their corresponding 2D base polygons.

To create these models we implemented a semi-automatic approach for modeling architecture from images [DTM96]. The user captures a number of photos of a building from different viewing positions, and creates a coarse model using simple base elements such as cubes or cylinders. In a second step, edges of the base model are linked to their corresponding edge



(a) (b) (c) (d)

Figure 7: The depth, albedo, specular, and normal component of each pixel for deferred shading (a). Cascaded shadow maps (b) and additional soft shadows and ambient occlusion effects (c). Final output image combining all shader effects (d).

in the input images. These links are used as input to an optimization process which adapts the shape of the building and the camera parameters in order to refine the coarse base model to a more faithful representation of the building. This model can then be textured using the original images as input (Fig. 6).

5 Visualization

The major challenge in visualizing the resulting huge city model in OpenGL [SA06] is the amount of data. Our final mesh consists of over 6 million unique vertices and 2.2 million triangles, textured using over 400 different materials. To render this amount of data at realtime framerates, one has to reduce the batch-count as much as possible [Wlo03]. We achieve this using a heavily optimized memory layout for the data and an Octree [JR00] that allows to cull nodes but still render all data in strictly sequential order with a minimum of drawcalls. Since most of the 400+ materials are combinations of background (wall) and foreground (windows) textures we dynamically combine multiple textures in a shader which reduces the texture count to 80. Reducing the memory footprint of the diffuse textures allows us to add additional visual effects using normal-maps and gloss-maps.

A further issue is the significant overdraw caused by the large number of occlusions which are inevitable in large city models. For overdrawn pixels, the complex shaders we utilize to enhance the visual quality are evaluated multiple times, leading to a significant performance reduction of the rendering system. We hence implemented the whole rendering pipeline using a deferred shading approach [Shi05]. In a first pass we render the scene into multiple framebuffers without complex lighting calculations or other visual effects. We end up with three framebuffers containing the depth, the albedo and specular level, and the normal direction of each pixel (Fig. 7 a).

In a second rendering pass these buffers are evaluated for each output pixel in order to restore the pixel's depth value and position for computing per-pixel lighting. This approach

Figure 8: Different views of the city of Aachen generated with our system.

guarantees that all computational expensive shader operations like multiple shadow look-ups, environment mapping and specular highlights are only computed at most once for each pixel on the screen. As a side note, however, it is important to take particular care of the reduced numerical precision during the rendering process. We solve this problem by using an adaptively computed near plane distance for different viewing situations, e.g., walking on the ground vs. flying over the city.

For the shadow generation we use multiple shadow maps [Ura05, For05]: two traditional shadow maps for high resolution shadows close to the viewer and low resolution shadows in the far distance (Fig. 7 b). These two shadow maps and the sky are separated on the screen using the stencil buffer. This prevents unnecessary shader computation in the sky and reduces branching in the shader on the ground, as for each stencil value the appropriate shadow map is constant [Per05]. Additionally we use a third shadow map which employs unsharp masking [LCD06] to simulate diffuse shadowing (Fig. 7 c). Additional basic effects like a simple animated sky and fog contribute greatly to a more realistic look, while environmental mapping, specular highlights and bloom give the scene a warmer appearance and lead to the final image of the scene [JR04] (Fig. 7 d). Despite the complexity of the overall scene and the different visual effects the render engine runs at more than 60 FPS on a standard PC with a NVIDIA Geforce 8800 graphics card.

Besides simple mouse and keyboard navigation, we implemented interaction metaphors for stereo visualization with optical head and motion tracking. In addition to standard fly- and walking-modes, we included a “giant”-mode where the user is enlarged with respect to the model. This allows to explore the city on a global level similar to scaled architectural models.

6 Conclusions

In this paper we have presented a complete, flexible and scalable pipeline for the efficient generation and real-time, high quality visualization of large-scale city models. We have identified four major subproblems (model generation from 2D map data, texture processing, image-based modeling for landmark buildings, and visualization of the resulting model) and developed efficient tools to solve each of them. The resulting system enables even non-expert users to rapidly create city models of high quality. While our main target application is the visualization of city models, the watertight models reconstructed with our method are well-suited for other applications like, e.g., simulations of radio wave propagation (Fig. 9).

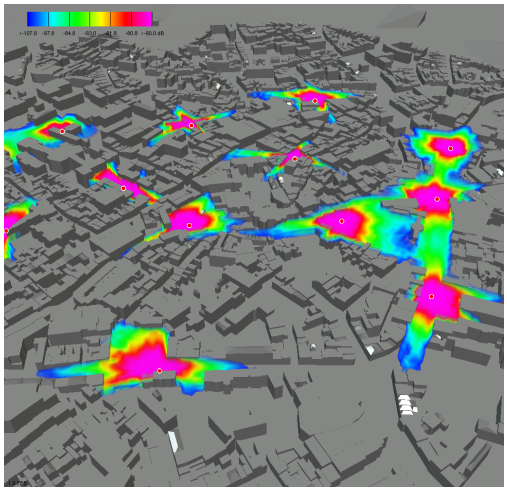


Figure 9: Simulation of GSM network field strength with the method of [SK06] using a watertight model of the city of Aachen reconstructed by our system.

One of the main advantages of our approach is the seamless transition from plausible city visualizations to fully correct city reconstructions: A plausible, high-quality model can be generated mostly automatic with only very little manual interaction and from quite rudimentary 2D base data. Our pipeline then allows for the easy generation and integration of correct textures for facades and detailed landmark buildings. While this is mostly relevant to applications targeting at a plausible visualization, the geometrical level of detail and correctness of the automatically generated model parts could be improved arbitrarily by integrating base data of higher quality for simulation purposes (e.g., using cadastral data including the number of floors for each building).

An interesting area for future work is the automation of the texture generation process. Currently the creation of textures and assignment to building facades requires a certain amount of manual interaction. With cameras mounted on a moving vehicle and a GPS system as well as image-based techniques for its localization, we aim at the rapid creation of large, correctly textured parts of the city model.

7 Acknowledgements

This project was a practical course at the Computer Graphics Group of RWTH Aachen University in the summer semester of 2007. The students which contributed to this project were (in alphabetical order): Gregor Fabritius, Lars Krecklau, Jan Kraßnigg, Christopher Manthei, Christian Martelock, Matthias Passon, Dominik Rausch, Tillmann Vogt and Melanie Winkler. We are particularly thankful to Dominik Rausch and Matthias Passon for additional modeling work.

References

[Ble08] Blender. http://www.blender.org/, June 2008.

[CAS77] S. Ishikawa C. Alexander and M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1977.

[CLCG06] N. Cornelis, B. Leibe, K. Cornelis, and L. Van Gool. 3D city modeling using cognitive loops. In *3DPVT '06*, 2006.

[DTM96] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96*, 1996.

[FJZ05] C. Früh, S. Jain, and A. Zakhor. Data processing algorithms for generating textured 3D building facade meshes from laser scans and camera images. *International Journal of Computer Vision*, 61(2):159–184, 2005.

[For05] T. Forsyth. Making shadow buffers robust using multiple dynamic frustums. In *Shader X4 Advanced Rendering Techniques*, chapter 4.5. Charles River Media, Boston, MA, USA, 2005.

[Geo08] Geobasis, Landesvermessungsamt NRW. http://www.lverma.nrw.de/, June 2008.

[Gis08] Open Source GIS. http://opensourcegis.org/, June 2008.

[Goo08] Google Earth. http://earth.google.com/, June 2008.

[GPSL03] S. Greuter, J. Parker, N. Stewart, and G. Leach. Real-time procedural generation of 'pseudo infinite' cities. In *GRAPHITE '03*, pages 87–ff, 2003.

[HZ04] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[JR00] G. James and J. Rorke. Loose octrees. In *Game Programming Gems*, chapter 4.11. Charles River Media, Boston, MA, USA, 2000.

[JR04] G. James and J. Rorke. Real time glow. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, chapter 21. Addison-Wesley Professional, Boston, MA, USA, 2004.

[LCD06] T. Luft, C. Colditz, and O. Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Trans. Graph.*, 25(3):1206–1213, 2006.

[LDG01] J. Legakis, J. Dorsey, and S. Gortler. Feature-based cellular texturing for architectural models. In *SIGGRAPH '01*, pages 309–316, 2001.

[LG06] M. Larive and V. Gaildrat. Wall grammar for building generation. In *GRAPHITE '06*, pages 429–437, 2006.

[MWH+06] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006.

[MZWG07] P. Müller, G. Zeng, P. Wonka, and L. Van Gool. Image-based procedural modeling of facades. *ACM Trans. Graph.*, 26(3):85, 2007.

[Ope08] Open Street Map. http://www.openstreetmap.org/, June 2008.

[Per05] E. Persson. Dynamic branching on non-ps3.0 hardware. In *Shader X4 Advanced Rendering Techniques*, chapter 8.4. Charles River Media, Boston, MA, USA, 2005.

[PM01] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *SIGGRAPH '01*, pages 301–308, 2001.

[PNF+08] M. Pollefeys, D. Nister, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, R. Yang, G. Welch, and H. Towles. Detailed real-time urban 3D reconstruction from video. *International Journal of Computer Vision*, 78:143 – 167, 2008.

[SA06] M. Segal and K. Akeley. The OpenGL Graphics System: A Specification (Version 2.1). http://www.opengl.org, 2006.

[Shi05] O. Shishkovtsov. Deferred shading in stalker. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter 9. Addison-Wesley Professional, Boston, MA, USA, 2005.

[SK06] A. Schmitz and L. Kobbelt. Real-time visualization of wave propagation. In *Proc. of MSPE*, pages 71–80. Gesellschaft für Informatik (GI), 2006.

[Sti75] G. Stiny. *Pictorial and Formal Aspects of Shape and Shape Grammars.* Birkhauser Verlag, Basel, 1975.

[SV01] I. Suveg and G. Vosselman. 3D building reconstruction by map based generation and evaluation of hypotheses. In *BMVC*, 2001.

[Ura05] Y. Uralsky. Efficient soft-edged shadows using pixel shader branching. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter 17. Addison-Wesley Professional, Boston, MA, USA, 2005.

[VD01] C. Vestri and F. Devernay. Using robust methods for automatic extraction of buildings. In *CVPR (1)*, 2001.

[Wlo03] M. Wloka. Batch, batch, batch: What does it really mean? http://developer.nvidia.com/docs/IO/8230/BatchBatchBatch.pdf, 2003.

[WMV+08] B. Watson, P. Müller, O. Veryovka, A. Fuller, P. Wonka, and C. Sexton. Procedural urban modeling in practice. *IEEE Computer Graphics and Applications*, 28(3):18–26, 2008.

[WWSR03] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Trans. Graph.*, 22(3):669–677, 2003.