

Programmable Graphics Processing Units for Urban Landscape Visualization

M. Kada, T. Balz, N. Haala, D. Fritsch

Institute for Photogrammetry, Universität Stuttgart, Germany

ABSTRACT: The availability of modern graphics cards allows high-end visualization by reasonably priced standard hardware. Real-time visualization of virtual 3D city models can therefore be provided area covering. Since also complex algorithms can be directly implemented in modern programmable graphics cards, their great computational power can be made available for a growing number of applications. As an example, techniques from photogrammetry and computer graphics are combined by a direct texture mapping of building façades from geo-referenced terrestrial images. In a second example, the graphics pipeline is adapted for the real-time simulation of SAR imagery by the modification of the imaging geometry from visual light to RADAR beams.

1 INTRODUCTION

Modeling and visualization of 3D urban landscapes has been of major interest in the past years. Nowadays, real-time applications are very popular, particularly if the data sets are streamed over the internet. Key markets for this kind of visualization are city planning and marketing, traffic management, three-dimensional car navigation, location-based services, entertainment and education. Although more abstract presentations based on generalized models (Kada 2006) have been proposed for applications with hand-held devices of limited screen size, a photorealistic visualization is in most cases preferred by the consumers.

Such real-time visualizations of artificial or real-world scenes are one of the main goals in computer graphics. Meanwhile, the applicability and accessibility of soft- and hardware tools developed in this field is further increased due to the evolution of the 3D graphics hardware from a fixed function to a programmable pipeline design. Thus, the application of graphics hardware is no longer limited to standard rendering tasks, but can be extended for complex algorithms. As it is demonstrated exemplarily in section 2, the graphics rendering pipeline can be used for the on-the-fly texture mapping from real world imagery. Within this implementation, a high level programming language is for example used for visibility analysis and the elimination of geometric effects like lens distortion. A modification of the standard geometric transformations for virtual image generation can also be used to allow for a real-time SAR simulation based on the graphics pipeline. While visualizations of 3D city and landscape models mainly aim on the generation of realistic scenes and animations e.g. for planning and navigation purposes, the implementation of SAR geometry, which is discussed in section 3 provides a key tools for SAR image understanding.

1.1 *Urban Landscape Database*

The 3D model of Stuttgart, which is used for our investigations, contains a digital terrain model of the city and the surrounding area of the size 50 * 50 km. The resolution of the DTM is

10 meter for the inner city region and 30 meter for the surrounding area. The included building models are provided by the City Surveying Office of Stuttgart. They were 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 (Wolf 1999). The resulting wireframe model contains the geometry of 36,000 buildings covering an area of 25 km², meaning that almost every building of the city and its suburbs is included. The overall complexity of all the building models 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.

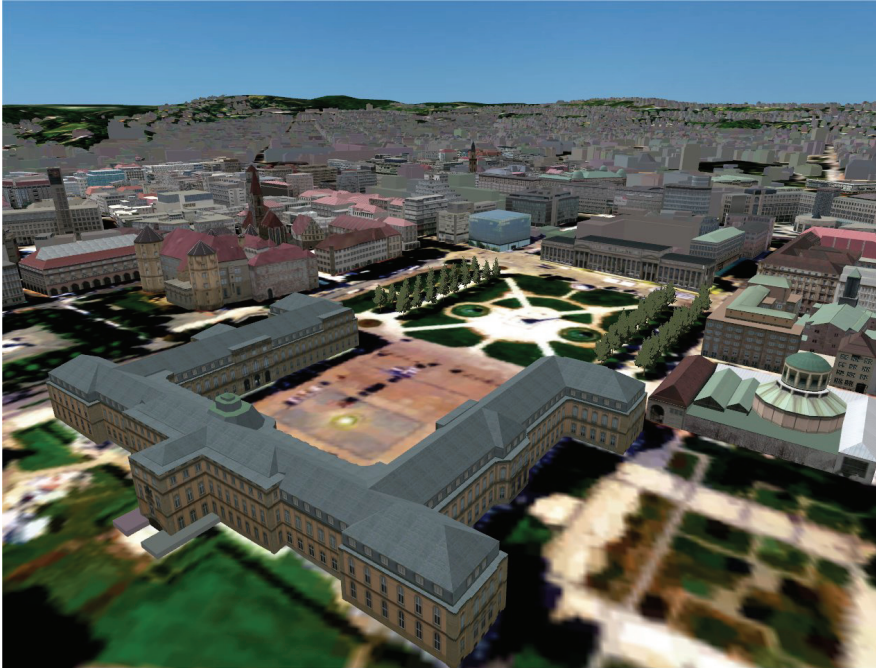


Figure 1. 3D city model of Stuttgart with terrestrially captured façade textures.

Airborne data collection efficiently provides complete sets of 3D building models at sufficient detail and accuracy for a number of applications. However, due to the viewpoint restrictions of airborne platforms, the collected building geometry mainly represents the footprints and the roof shapes, while information for the facades of the buildings is missing. Thus, in order to allow for high quality visualizations from pedestrian viewpoints, texture mapping based on terrestrial images is usually applied, additionally. To improve the visual appearance, the façade textures of 1,000 buildings located in the main pedestrian area were captured. For this purpose, approximately 8,000 ground based close-up photographs of the building façades were taken using a standard digital camera. The textures were extracted from the images, perspective corrected, rectified and manually mapped on the corresponding planar façade segments. We managed to process the aforementioned 1,000 buildings in roughly 30 man-months. Because of the large size of the original texture dataset, we had to down-sample the textures to a resolution of approximately 15 centimeters per pixel. Buildings with no real captured façade textures were finally colored randomly with different colors for the façade and the roof. An exemplary visualization of this data set is given in Figure 1.

1.2 Real-time Visualization

The visualization of this 3D city model data set was implemented using the terrain rendering library libMini which is licensed under an open source license. The approach that is realized in the library recursively generates triangle fans from a view-dependent quad-tree structured triangulation. It is very easy to integrate the library in other software packages as the API is simple to use. To suppress popping artifacts that can be otherwise experienced because of changes in geometry, a technique called geomorphing is applied which slowly moves newly introduced

vertices from a position on the terrain to its final position. The 3D building models are pre-processed for visualization in order to avoid unnecessary state changes in the rendering pipeline. All buildings are pre-transformed to lie in the same coordinate system and buildings without textures are grouped together to form larger data units. Primitives are rendered as indexed vertices in a triangle list. As the position of the global light source does usually not change, we omitted the normal vector for each vertex and pre-lit all vertices. The memory requirement of the vector data could be reduced this way by 42%.

The performance analysis of the visualization application has been conducted on a standard PC equipped with a 3.0 GHz Intel Pentium 4 processor, 1 GB RAM and an ATI X800 Pro compliant graphics card. In previous work we had used the impostor technique to accelerate the rendering of the building models (Kada et al 2003). As performance has considerably increased with the latest hardware generations, we felt that the speed-up of the impostor approach does not justify its disadvantages (especially the occasional frame rate drops) anymore. Instead we preprocessed the data so that most of the models and textures were guaranteed to remain on the graphics memory. The time extensive paging of data in and out of dedicated graphics memory was consequently minimized. Running at a screen resolution of 1280*1024 the application almost reaches real-time performance, meaning that approximately 15 to 20 frames per second are rendered.

2 PROGRAMMABLE GRAPHICS HARDWARE FOR DIRECT TEXTURE MAPPING

During visualization, the façade texture is provided by linking the 3D object coordinates of the building models to the corresponding coordinates of the texture images. The required world to image transform can be provided easily, if the exterior orientation and the camera parameters are available for the terrestrial images. However, if these images are directly used, the quality of visual realism is usually limited. This results from the fact that standard texture mapping only allows for simple transformations, while complex geometric image transformations to model perspective rectification or lens distortion are not available. Thus, these effects are usually eliminated before texture mapping by the generation of ‘ideal’ images, which are then used as an input for rendering. Additional pre-processing steps are required for the definition of occluded parts or the selection of optimal texture if multiple images are available.

Alternatively, these tasks can be processed on-the-fly by programmable graphics hardware. Such an implementation can additionally help to solve problems resulting from self-occlusions to integrate multiple images during texture mapping. By these means, the façade texture is directly extracted from the original images; no intermediate images have to be generated and stored. Thus, within the whole process image pixels are interpolated only once, which results in façade textures of higher quality. Since our implementation for the on-the-fly generation of façade texture is based on technologies that can be found in today’s commodity 3D graphics hardware, thus the computational power of such devices can be used efficiently.

Graphics processing units (GPU) that are integrated in modern graphics cards are optimized for the transformation of vertices and the processing of pixel data. As they have evolved from a fixed function to a programmable pipeline design, they can now be utilized for various fields of applications. The programs that are executed on the hardware are called shaders. They can be implemented using high level programming languages like HLSL (developed by Microsoft) (Gray 2003) or C for graphics (developed by NVIDIA) (Fernando 2003). In our approach shaders are used to realize specialized projective texture lookups, depth buffer algorithms and an on-the-fly removal of lens distortions for calibrated cameras. This approach can be implemented based on the graphics API Direct3D 9.0 which defines dynamic flow control in Pixel Shader 3.0. By these means, the lens distortion in the images can be corrected on-the-fly in the pixel shader.

2.1 *Texture Extraction and Placement*

Our approach uses the graphics rendering pipeline of the graphics card to generate quadrilateral texture images. In general, the function of the pipeline is to render a visualization of a scene from a given viewpoint based on three-dimensional objects, textures and light sources. Because the texture images, which are mapped against the façades during visualization, have to

be represented by quadrilaterals, the polygons of the building are substituted by their bounding rectangles during the extraction process (see Figure 2 left). For these bounding rectangles 3D world coordinates are available. This information is used to calculate the corresponding image pixels, which provide the required façade texture. In more detail, the first step is to set up the graphics rendering pipeline to draw the entire target pixel buffer of the final façade texture. For this purpose, the transformation matrices are initialized with the identity, so that drawing a unit square will render all pixels in the target buffer as wanted. As no color information is provided yet, a photograph must be assigned to the pipeline as an input texture from where to take the color information from. As mentioned above, the polygon's projected bounding box defines the pixels to be extracted from the input texture. So in addition to the vertices, the texture coordinates of the four vertices of the unit square are specified as the four-element (homogenous) world space coordinates of the bounding box. Setting the texture transformation matrix with the aforementioned transformation from world to image space concludes the initialization.

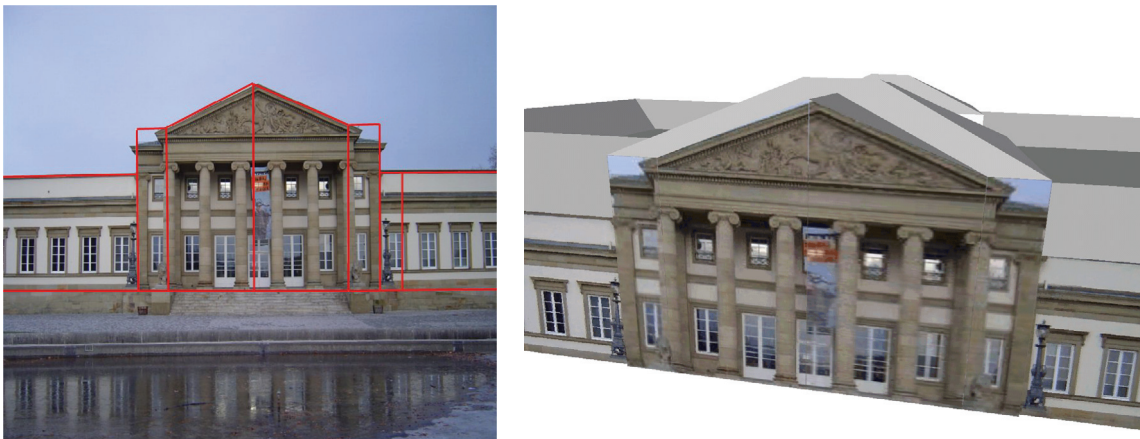


Figure 2: Projected 3D building model overlaid on the input photograph

During rendering, the rasterizer of the GPU linearly interpolates the four-dimensional texture coordinates across the quadrilateral. A perspective texture lookup in the pixel shader results in the perspective correct façade texture (see Figure 2 right). After the extraction, the textures need to be placed on the corresponding polygons. In order to find the two-dimensional texture coordinates for the polygon vertices, a function identical to `glTexGen` ((Shreiner, Woo, Neider 2003)) of OpenGL is used.

2.2 Image fusion



Figure 3: Original input image and textured building model with occlusion culling disabled.

A common problem is that parts of the building façades are not visible in the photograph due to self-occlusions. If this effect is not modeled correctly erroneous pixels are extracted from the respective texture image (see Figure 3). To avoid such artifacts, invalid pixels that belong to

other polygons must be identified and marked. By using the depth buffer algorithm, the closest polygon for each pixel in the photograph can be determined. We use a pixel shader to calculate the depth value and render it directly into a 32 bit floating-point depth texture. During texture extraction, this value is then read out in the extracting pixel shader using the same texture coordinates as for the color lookup. After the perspective divide is applied to the texture coordinates, the z-component holds the depth value for the current polygon pixel. A comparison of these two depth values then determines if the pixel in the color value belongs to the polygon.

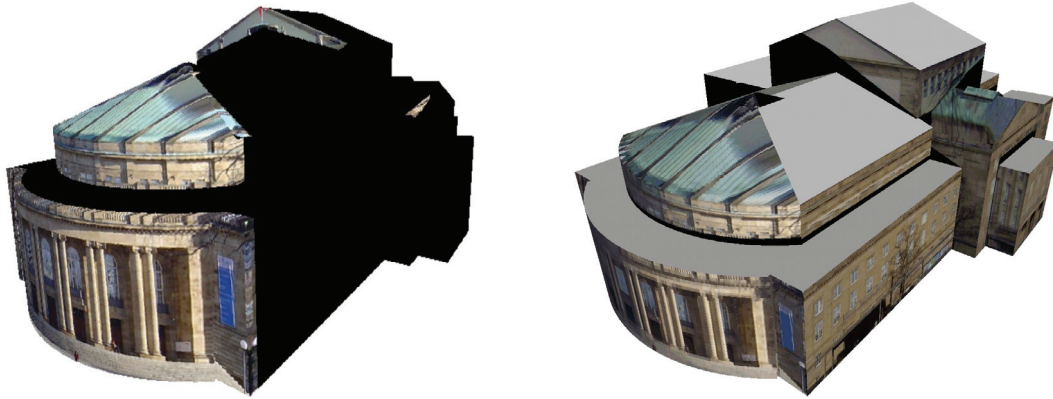


Figure 4: Textured building model, occlusion culling enabled (left) and mapping from multiple images with occlusion free pixels selected (right).

Figure 4 (left) exemplarily shows the result of texture mapping where occluded pixel values have been masked out. When processing more than one photograph, the final color can be merged by using the closest, non-occluded pixel in all images. Even though the approach is brute force, it is still very efficient with hardware support. If texture image information from various positions is available the final color can be merged by using the closest, non-occluded pixel in all images to generate the final façade texture (Figure 4 right).

The texture extraction algorithm was implemented in C++ with the graphics API Direct3D 9.0, which includes the high level shading language HLSL. The performance analysis has been conducted on a standard PC with an Intel 4 3.0 GHz Processor, 1GB of DDR-RAM and a graphics card that is based on the ATI 9800 graphics processing unit (GPU) with 256 MB of graphics memory. With all features enabled, the extraction time for a model of approx. 146 polygons and 16 input images is still below one second.

3 REAL-TIME SAR SIMULATION OF URBAN SCENES

Image rendering usually applies a perspective projection of 3D models for scene generation. However, programmable graphics cards can also be used to simulate scenes, which are depicted by microwaves instead of visual light. In reality, SAR images are acquired based on the back-scattered energy of emitted RADAR beams. Compared to optical remote sensing, SAR is advantageous due to the range and weather independence of the image acquisition process. However, problems result from the fact, that the interpretation of SAR imagery can become very complex especially for three-dimensionally structured scenes. For this reason, the analysis and interpretation of SAR data is supported frequently by SAR simulators (Leberl & Bolter 2001). Especially for dense urban environments, such simulators are also required during mission planning to avoid occluded areas (Soergel et al 2003). Since the interaction of different SAR effects is very difficult to understand even for experienced human operators, such simulators are also used frequently for training purposes. As it is demonstrated in Figure 5, such a tool can for example combine the simulated SAR image with a 3D visualization of the respective 3D building objects (Balz 2003). If real-time SAR simulation is feasible, an interactive change of the building shapes by the operator will result in the adaption of both the 3D presentation of the buildings and the simulated SAR image.

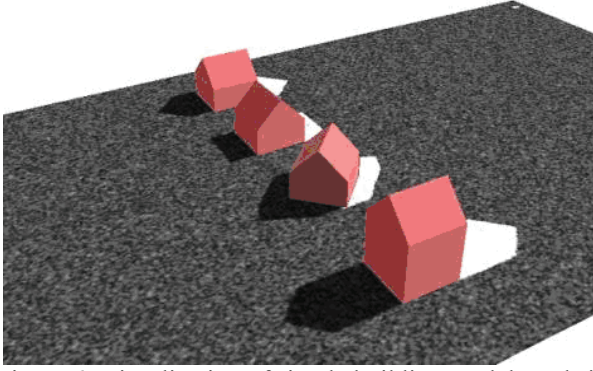


Figure 6: Visualization of simple building models and simulated SAR image

Usually, SAR image simulation is realized based on ray-tracing, i.e. each pixel requires the computation of one ray. Since this approach is rather computational time consuming, we adapt the standard graphics rendering pipeline to compute the geometry and radiometry of SAR scenes in real-time. Our SAR simulator SARViz was implemented using Microsoft's DirectX API and the High-Level Shading Language (HLSL) for GPU programming (Balz 2006). SARViz uses slightly modified DirectX meshes as input source, which can be generated by a variety of 3D CAD tools. Alternatively, data sources like Arc-GIS datasets can be imported. The distance geometry of a SAR image is implemented in the vertex shader, which is a highly specialized part of a graphic card, optimized for matrix calculations. Vertex shaders transform each point from the model coordinate system to world coordinates and then subsequently to screen or image coordinates. Since the GPU, which is offered by modern graphic cards, is a data-parallel streaming processor working in a single-instruction, multiple data (SIMD) fashion, a massively parallel computing is feasible. After each triangle is processed and rasterized, the resulting pixels are processed by the pixel shader to compute the corresponding radiometry. Modern pixel shaders (pixel shader version 3.0 or higher) support branching and more than 512 machine-code commands, which is sufficient even for complex tasks like SAR simulation.

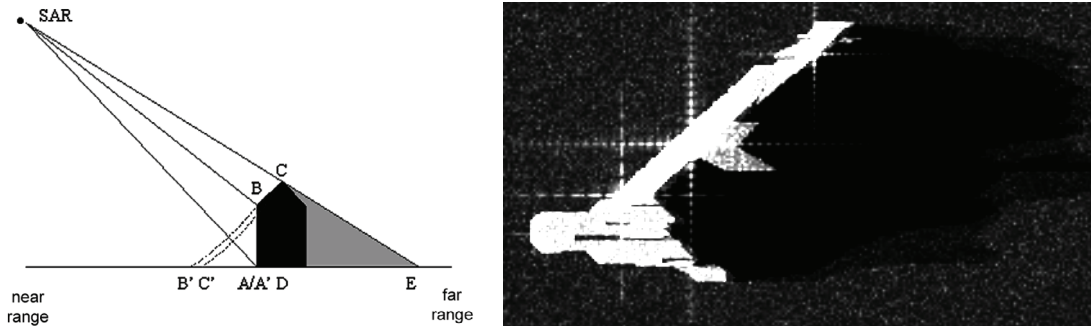


Figure 7: SAR imaging geometry in range (left) and high resolution SAR simulation of the "Stiftskirche" in Stuttgart (right)

Compared to optical systems SAR images are not only based on different wavelengths, but are subject to a totally different imaging geometry. As demonstrated in Figure 7, this so-called distance geometry of SAR images results in pixel positions, which depend on the distance between the sensor and the respective object. Thus, high object points, which have a relatively short distance to the sensor is mapped closer to near-range than low points. This well-known foreshortening effect is clearly visible in the high resolution SAR simulation of the "Stiftskirche" depicted in the right image of Figure 7. SAR radiometry depends on the backscattered energy of the RADAR beams. Based on given material properties, like the dielectric constant, the surface normals of the respective object patches, as well as the sensor properties, these reflection strengths can be calculated in the pixel shader. However, SAR images cover a high data range which cannot be represented by 8-bit images. To cover the complete dynamic range, 32-bit data is adequate. Modern graphic cards support high-dynamic range lighting with 32-bit floating point accuracy, which is sufficient for the simulation of SAR images. To use this abil-

ity, the pixel shader first renders to a 32-bit floating point texture. In the next step this texture is enhanced for 8-bit display and rendered to the screen, enabling the SARViz simulator to deliver both 8-bit data for visualization, but also 32-bit floating point images for further processing.

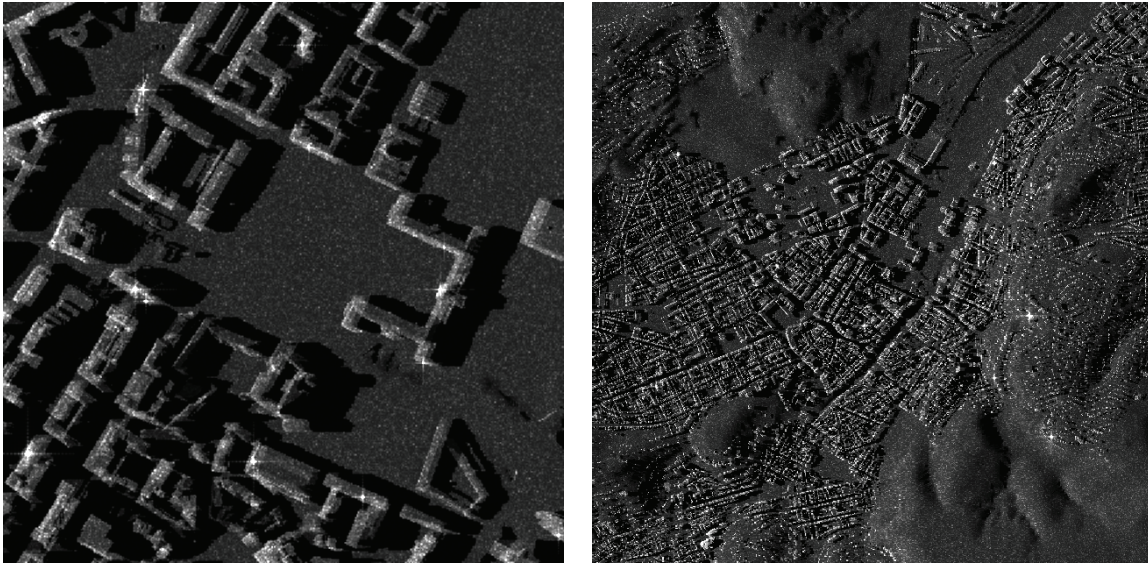


Figure 8: SARViz simulation of area surrounding the “Neue Schloss” in Stuttgart (left) and of the complete 3D city model of Stuttgart (right).

The SARViz simulator is designed to simulate complex and extensive 3D models. Complex urban environments like they are depicted in the exemplary simulations in Figure 8 are simulated in about 50 milliseconds using a NVIDIA GeForce 7900 GTX graphic card without the need for model simplification. This time does not include the time needed to copy the data to the video memory. The use of simplified models could allow even higher frame rates during the low-resolution simulation of complex models.

4 CONCLUSION

Modern graphic cards enable high-end visualization using reasonably priced standard hardware. Also due to this fact, real-time visualization of virtual 3D city models is now integrated in a growing number of applications. In addition to the great computational power of these systems, programmable graphics hardware also allows the direct implementation of complex algorithms. The direct texture mapping of building façades as well as the real-time SAR simulation demonstrated that ‘photogrammetric’ tasks like the transformation and mapping of world and image coordinates can be directly integrated in the graphics rendering pipeline. This allows for time efficient solutions, which profit from the tremendous increase in computational speed, which is and will be provided by consumer type graphic cards.

5 REFERENCES

- Balz, T. [2006]. Real-Time SAR Simulation on Graphics Processing Units. 6th European Conference on Synthetic Aperture Radar (EUSAR 2006)
- Fernando, R.a.K.M. [2003]. The Cg Tutorial. Addison-Wesley
- Gray, K. [2003]. The Microsoft DirectX 9 Programmable Graphics Pipeline. Microsoft Press.
- Kada, M. [2006]. 3D building generalization based on half space modelling. Joint ISPRS Workshop on Multiple Representation and Interoperability of Spatial Data, pp.6 pages, on CD.
- Kada, M., Roettger, S., Weiss, K., Ertl, T. & Fritsch, D. [2003]. Real-Time Visualisation of Urban Landscapes Using Open-Source Software. ACRS 2003 ISRS, Busan, Korea.
- Leberl, F. & Bolter, R. [2001]. Building reconstruction from Synthetic Aperture Radar images and interferometry. Automatic Extraction of Man-Made Objects From Aerial and Space Images (III).
- Shreiner, D., Woo, M. & Neider, J. [2003]. OpenGL Programming Guide. Addison-Wesley

Soergel, U., Schulz, K., Thoennesen, U. & Stilla, U. [2003]. Event-driven SAR Data Acquisition in Urban Areas Using GIS. GIS 16(12)

Wolf, M. [1999]. Photogrammetric Data Capture and Calculation for 3D City Models. Photogrammetric Week '99, pp.305-312.