Accelerating Radio Wave Propagation Predictions by Implementation on Graphics Hardware

Daniel Catrein, Michael Reyer, Tobias Rick
Institute of Theoretical Information Technology
RWTH Aachen University
D-52056 Aachen, Germany
Email: {catrein,reyer,rick}@ti.rwth-aachen.de

Abstract— **Fast radio wave propagation predictions are of tremendous interest, e.g., for planning and optimization of cellular radio networks. We propose the use of ordinary graphics cards and specialized algorithms to achieve extremely fast predictions. Our implementation of the empirical COST-Walfisch-Ikegami model allows the computation of several hundred predictions in one second in a 7 km^2 urban area. Further, we present a ray-optical approach exploiting the programming model of graphics cards. This algorithm combines fast computation times with high accuracy.**

I. Introduction

Radio wave propagation models play an essential role in planning, analysis and optimization of radio networks. For instance, coverage and interference estimates of network configurations are based on field strength predictions. For network planning, a vast amount of different configurations has to be explored to achieve optimal utilization of radio resources, demanding for extremely fast radio wave propagation algorithms.

An overview of radio wave propagation models is given in [1] and [2]. Approaches for estimating field strengths can basically be divided into (semi-)empirical and ray-optical models. The semi-empirical COST-Walfisch-Ikegami model [1] estimates the received power predominantly on the basis of frequency and distance to the transmitter. Ray-optical approaches identify ray paths through the scene, based on wave guiding effects like reflection and diffraction. Semi-empirical algorithms usually offer fast computation times but suffer from inherent low prediction quality. Ray-optical algorithms feature a higher prediction quality at the cost of higher computation times. For comparison with our new algorithms we use a fast and well tested ray-optical algorithm (CORLA) [3], [4], in this paper. Current work on prediction algorithms which are based on ray-optical approaches can be found in [5] and [6].

Modern graphics cards offer tremendous computing power due to their highly parallel architecture. Additionally, the performance of graphics cards doubles every half year [7], compared to the performance of standard CPU's which increases with the factor $\sqrt{2}$ every year, according to Moore's Law. Thus, making the graphics card an attractive platform for computation-intensive tasks. The computational power offered by graphics cards is already exploited for problems that go beyond graphical applications, like sorting or physical simulations. Implementations on the *Graphics Processing Unit* (GPU) often accelerate algorithms by over an order of magnitude compared to the standard CPU implementation. An overview on some ideas of *General-Purpose Computations on GPUs* (GPGPU) is presented in [7]. Recent work includes the mapping of classical ray tracing programs to the GPU, [8] and [9], which is of particular interest with regard to ray-optical wave propagation algorithms.

In this paper, we show that the use of graphics hardware allows the acceleration of existing approaches, if mapped correctly to the graphics programming concept. The algorithm developed in Section V extends the ray launching approach of [4]. However, the present algorithm is completely redesigned to benefit from graphics hardware. The algorithm adapts shadow algorithms from computer graphics that run extremely fast on graphics hardware to compute ray paths based on roof diffraction.

This paper is organized as follows. In Section II we briefly introduce the underlying programming paradigm of today's graphics cards. The semi-empirical COST-Walfisch-Ikegami model is used as a kind of prototype to show the abilities of graphics hardware based implementations in Section III. Section IV presents a path loss model that derives the received power based on distance and diffraction over rooftops. Section V describes an algorithm that calculates roof diffraction ray paths. This algorithm is explicitly developed to run on graphics hardware. Section VI discusses the introduced algorithms, with respect to prediction quality and computation time. We conclude this work with an overview of the results in Section VII.

II. Graphics Hardware

The underlying architecture of graphics cards is called *Single Instruction Multiple Data* (SIMD), i.e., many parallel processors execute the same instructions at a time on different parts of data. In addition to the high computing power, modern graphics cards are programmable at certain stages of their *Rendering Pipeline* (Figure 1).

The pipeline consists of an input, a processing and an output unit. The input consists of planar geometric objects, e.g., triangles or quadrangles, described by three dimensional coordinates (*vertices*) with connectivity information and arbitrary numerical information (*textures*). In the first pipeline step multiple vertex processors execute in parallel the instructions

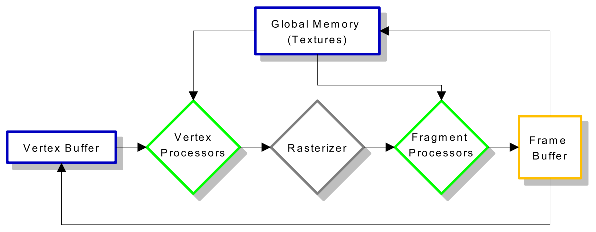


Fig. 1. The Graphics Rendering Pipeline.

from a user-written program (*kernel*) on the vertices. Usually, geometric transformations like translations and rotations are applied.

In the next step, the processed geometric objects are rasterized into discrete points (*fragments*). Each fragment has a screen position (*pixel position*), a *depth value* and additional numerical information.

Analogous to the vertex processors, multiple fragment processors execute user-written programs on each fragment in parallel, producing the final result of the computation. Usually, the output consists of a vector $v \in \mathbb{R}^3$ which is commonly interpreted as color information.

In a final non-programmable stage all fragments are collected and recorded in the *framebuffer*. If multiple fragments are mapped to the same pixel position, the *depth test* decides which one is written into the framebuffer, by comparing the fragments' depth values.

Both, vertex and fragment processors can be programmed in a slightly restricted C-like language. The major drawback of the GPU programming model is that each vertex or fragment is processed independently, without access to others. Only the non-programmable depth test at the very end of the pipeline may compare information of several fragments on the same position.

III. Semi-empirical Model Evaluation on the GPU

To proof the power of GPUs for radio wave propagation predictions we implemented the COST-Walfisch-Ikegami (COST-WI) model [1]. In this model, the path loss at a receiver point r is given by

$$P^{\text{dB}}_{\text{COSTWI}}(r) = \begin{cases} P^{\text{dB}}_{\text{LOS}}(r) & , r \text{ in line-of-sight} \\ P^{\text{dB}}_{\text{NLOS}}(r) & , \text{otherwise}, \end{cases} \quad (1)$$

where $P^{\text{dB}}_{\text{LOS}}(r)$ and $P^{\text{dB}}_{\text{NLOS}}(r)$ are functions of the distance between transmitter and receiver and some transmitter specific parameters as frequency and height. As in [10], we neglect the parameter containing the orientation of the road. Furthermore, we assume a flat receiver plane and 2.5 dimensional building data, i.e., each building is described by its polygonal outline and a single height. To evaluate the COST-WI model, it has to be determined whether a given receiver point lies in line-of-sight to the transmitter. Afterwards, we have to calculate the distance to the transmitter and evaluate (1).

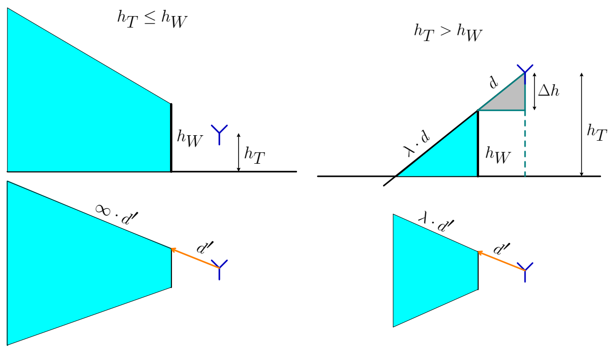


Fig. 2. Non-line-of-sight region of a transmitter beneath (left) and above (right) wall height, side view (upper) and top view (lower).

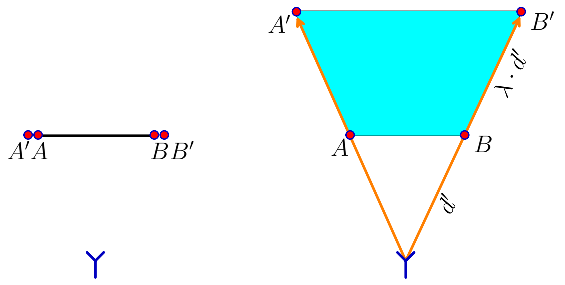


Fig. 3. Vertex processing.

The line-of-sight calculation is based on shadow algorithms from computer graphics, which can be implemented on recent graphics hardware. Our method is based on the so-called *shadow volumes* [11]. The basic idea of this technique is to construct a polygonal representation of the shadow cone. We call the intersection of the shadow cone and the receiver plane the *shadow polygon*. Regions are in line-of-sight, if they are in no shadow polygon and vice versa. The shadow cones are constructed by identifying the silhouette edges of the shadow caster and by moving them away from the light source (here, the transmitter).

The construction of the shadow polygon for a single wall is depicted in Figure 2. The shadow polygon is a quadrangle, where two of its corners are the end points of the wall. The remaining two corners are given by the intersection of the receiver plane and the straight lines trough the transmitter and the wall points. If the transmitter is located above the top of the wall, $\Delta h = h_T - h_W > 0$, the shadow polygon is finite and its dilation λ can be calculated by

$$\frac{h_W}{\Delta h} = \frac{\lambda \cdot d}{d} \qquad \Longrightarrow \qquad \lambda = \frac{h_W}{\Delta h}.$$

Otherwise the shadow polygon extends to infinity, in this case let $\lambda = \infty$.

This method can be implemented efficiently on graphics

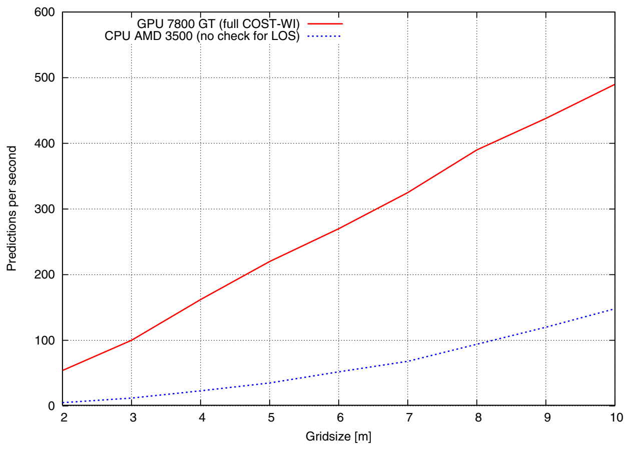


Fig. 4. CPU vs. GPU COST-WI implementation (Runtimes from the COST-Munich scenario).

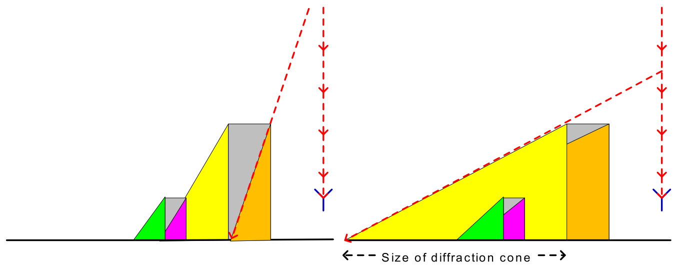
hardware. We decompose the building data into single walls. Let $A = A'$ and $B = B'$ denote the two end points of a wall and their duplicates, see Figure 3. We use the degenerated quadrangle $A'ABB'$ as input for the rendering pipeline. The vertex processors shift A' and B' away from the transmitter by the corresponding λ, resulting in the outline of the shadow polygon. The rasterizer discretizes the supplying area into fragments, and fills fragments inside the shadow polygon with non-line-of-sight information. The fragment processors execute a program to evaluate the path loss formula (1).

We implemented our algorithm in OpenGL for recent GPUs. For comparison we implemented just the evaluation of $P^{\text{dB}}_{\text{LOS}}$ on a CPU. Note, we do not check if a point is in line-of-sight on the CPU, which would presumably be the most expensive part of a full COST-WI implementation. Furthermore, evaluating $P^{\text{dB}}_{\text{LOS}}$ is computationally less expensive than evaluating $P^{\text{dB}}_{\text{NLOS}}$. Figure 4 shows that our GPU implementation of the full model clearly outperforms even the simplified CPU implementation. We tested our algorithm on the COST-Munich data [12], an area of 2.4×3.4 km^2. Figure 4 shows the number of predictions per second for this area depending on the resolution from 2 m to 10 m.

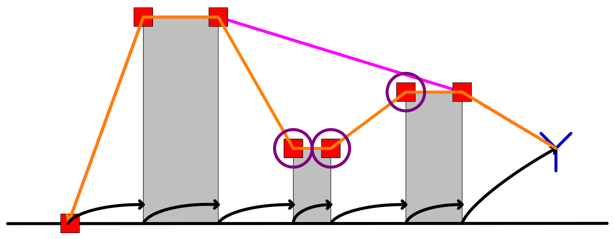
A CPU implementation of the full COST-WI model in a commercial tool like Winprop needs "less than one minute" for a single prediction with $72 \cdot 10^3$ evaluations, see [10]. We achieve more than fifty predictions per second, even with a resolution of 2 m on the COST-Munich scenario, which corresponds to more than $2 \cdot 10^6$ evaluations of (1) per prediction. Therefore, the implementation on GPUs allows for real-time prediction for a moving transmitter.

IV. Roof Diffraction Model

In this section we introduce the *Roof Diffraction Model* (RDM) for urban environments, see [4]. We assume that rays propagate in a straight line from the transmitter and may be diffracted downwards at the roof of buildings. The path loss



(a) Creating the Diffraction Wall Map.



(b) Creating the convex ray path.

Fig. 5. Principles of the ray path calculation.

for a receiver point r is modeled by

$$P^{\text{dB}}_{\text{RDM}}(r) = c_f + \gamma \cdot 10 \cdot \lg(d_r) + \sum_{i=1}^{K_r} g\left(\alpha^i_r\right), \tag{2}$$

with the well-known frequency dependent term

$$c_f = 20 \lg\left(\frac{4\pi f}{c}\right),$$

where c denotes the speed of light. The second summand depends on the path loss exponent $\gamma > 0$ and the length of the diffracted path d_r. K_r denotes the number of roof diffractions and α^i_r is the i-th diffraction angle. The function

$$g(\alpha) = b_0 + b_1\alpha + b_2\alpha^2, \quad \alpha \in \left[0, \frac{\pi}{2}\right],$$

with parameters $b_0, b_1, b_2 \in \mathbb{R}$ models the attenuation due to a diffraction angle α.

Adequate values for the parameter γ and the coefficients b_0, b_1, b_2 are retrieved from a calibration with measurement data. Route METRO202 of the COST-Munich scenario yields

$$\gamma = 2.76; \quad b_0 = 3.37; \quad b_1 = 0; \quad b_2 = 4.9.$$

The above values are used in this paper when calculating the path loss.

V. Roof Diffraction on the GPU

In this section we will extend the ideas presented in Section III to calculate the diffracted ray path for the RDM model. First we explain the principles of our algorithm, briefly. For each point in the receiver plane we determine if it is in line-of-sight to the last diffracting wall on the ray path to the transmitter. This information is written into the *Diffraction Wall Map* (DWM). We use it to calculate the last diffraction point for each receiver point, i.e., the intersection of the ray

Algorithm 1 BUILDDIFFRACTIONWALLMAP(WallSet $\mathcal{W}$, Transmitter T)

```
CLEARBUFFER(Framebuffer,RGBA(0,0,0,0))
CLEARBUFFER(Depthbuffer,0)
$h_T$ ← T.height
T.height ← STARTHEIGHT()  {Start at almost infinity.}
while T.height $\geq h_T$ do
  for all wall $\in \mathcal{W}$ do
    {In the vertex processor}
    poly ← GETSHADOWPOLYGON(wall,T)
    {In the rasterizer}
    fragments ← RASTERIZEPOLYGON(poly)
    {In the fragment processor}
    for all frag $\in$ fragments do
      frag.color ← wall.ID
      frag.depth ← T.height
      {Execute depth test}
      if Depthbuffer[frag.pos] $<$ frag.depth then
        Depthbuffer[frag.pos] ← frag.depth
        Framebuffer[frag.pos] ← frag.color
      end if
    end for
  end for
  T.height ← DECREASEHEIGHT(T.height)
end while
return Framebuffer
```

path and the last diffracting wall. This information is stored in the *Diffraction Point Map* (DPM). For each receiver point we traverse the DPM until we reach the transmitter to generate a sequence of points containing all possible diffraction points, the *Diffraction Sequence Map* (DSM). Exploiting the height information at those points the diffracted ray path is reconstructed.

To obtain the DWM containing the last diffracting wall for each point in the receiver plane, we use the procedure sketched in Figure 5(a). We let the transmitter "drop" from an infinite height down to its original height. For each height of the transmitter we use the shadow polygon algorithm described in Section III. We mark each shadow polygon with the shadow casting wall's unique ID. These IDs are written in the DWM, once a point enters shadow for the first time.

This method can be implemented efficiently on graphics card, only. The implementation is sketched in Algorithm 1. We use the depth buffer to ensure, that only the very first shadow casting wall is recorded. Implementing the algorithm requires to choose discrete heights while dropping the transmitter. As the evaluation of a single height is very fast, a high height resolution can be chosen without slowing down the whole algorithm significantly, see Section VI.

The DPM is calculated in the fragment processors once we have obtained the DWM. For every receiver point the diffracting wall is stored in the DWM. The last diffraction point can be found easily, by intersecting the corresponding wall segment with the straight connection between the receiver

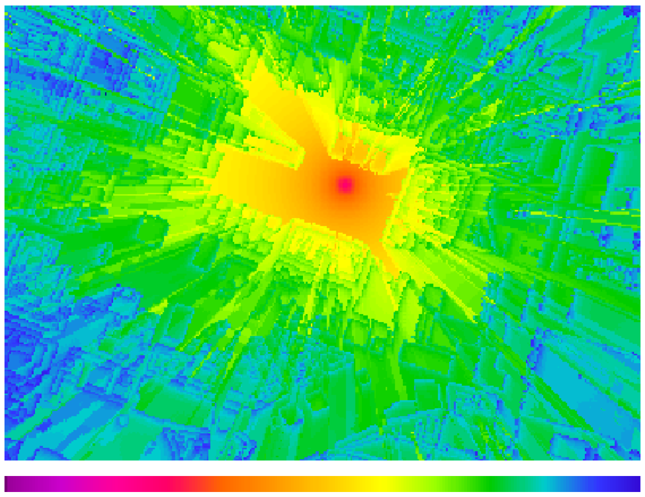


Fig. 6. Field strength prediction of the Roof Diffraction Model in Munich.

Task / Runtime	GPU Nvidia 7800 GT, CPU AMD 3500
Diffraction Wall Map	0.05s (GPU)
Diffraction Point and Sequence Map	0.8s (GPU)
Convex ray path and model evaluation	2.2s(CPU)
Total	**3.05s (GPU+CPU)**
CORLA (only roof diffraction, recursion depth of 6)	9s (CPU)

TABLE I
RUNTIME ANALYSIS OF THE MUNICH-SCENARIO
(7 KM^2 AT A RESOLUTION OF FIVE METER).

point and the transmitter, with its height given by the wall height. In this step, some effort is necessary to cope with discretization effects.

We traverse the DPM to generate the DSM in the following way. Starting from a receiver point we look up its diffraction point in the DPM. For this diffraction point we continue the lookup until we reach the transmitter, as depicted in Figure 5(b). Note, this sequence contains all diffraction points on the ray path, but it may also contain further points. It can be shown, that the ray path must be convex. Further more, only two dimensions have to be considered, as all points lie in a plane. Therefore, we use Andrew's Monotone Chain algorithm [13] to construct the final convex ray path. As the points on the path are already sorted, this step can be skipped here.

Now we have all the information to evaluate the path loss from (2). Both the convex ray path calculation and the path loss evaluation are currently implemented on the CPU.

VI. Comparisons

For benchmarking we used the building and measurement data released in the COST 231 action for the city of Munich [12]. All predictions were performed on the whole supplied area, of approximately 7 km^2 with a resolution of 5 m. Figure 6 visualizes a RDM prediction for a part of this area. Table I shows the runtime of the different parts of

Prediction model	mean error	MSE	std. dev.	Runtime
Ericsson (Raytracing + COST-WI), [1]	1.4 dB	-	7.5 dB	-
Uni.-Karlsruhe (Raytracing), [1]	−1.0 dB	-	8.6 dB	-
CORLA	0.1 dB	5.7 dB	5.7 dB	23 s
CORLA (roof diffraction only)	0.3 dB	7.3 dB	7.3 dB	9 s
COST-WI (GPU)	**11.6dB**	**14 dB**	**7.9dB**	**0.0045 s**
RDM (GPU)	**0.3 dB**	**7.4 dB**	**7.4 dB**	**3.05 s**

TABLE II

ACCURACY OF PROPAGATION MODELS IN THE COST-MUNICH SCENARIO ALONG ROUTE METRO202.

our roof diffraction algorithm. Obviously, constructing the diffraction wall map is extremely fast, consuming less than 2% of the total runtime of the algorithm. Calculating the diffraction point and sequence map take about 25% of the overall time, however, most of the time is currently spend to cope with discretization effects. We expect to reduce this time significantly in the future. Calculating the convex ray path is currently the most expensive part, last but not least because it is implemented on the CPU. This problem will be addressed in the future, too. However, even our current version is three times faster compared to roof diffraction calculated by the highly optimized CPU-only implementation CORLA [4].

Table II compares the accuracy of both, our COST-WI and RDM implementation with the accuracy of CORLA and results published in [1]. Even though the RDM model considers roof diffraction only, the mean squared error (MSE) between prediction and measurement is as low as 7.3 dB on measurement route METRO202 with a mean error of 0.3 dB. Figure 7 visualizes measurement and prediction with RDM on route METRO201, on which a MSE of 4.7 dB is achieved.

VII. CONCLUSIONS

In this paper we demonstrated the value of graphics cards as means of accelerating the evaluation of radio wave propagation models. The parallel architecture of today's graphics cards is exploited to achieve extremely fast computation times. We show that empirical models like the COST-WI model can be evaluated extremely fast. About 200 predictions per second can be achieved on recent hardware.

Implementing the more sophisticated roof diffraction model partially on graphics hardware results in fast and accurate predictions. This proofs the potential of using GPUs for field strength predictions. However, as seen in Section V, adapting existing algorithms to the programming concept of graphics cards is a challenging task.

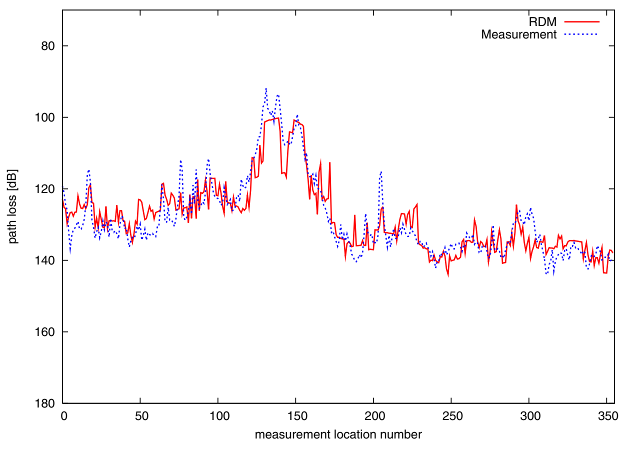


Fig. 7. Comparison between measured and predicted path loss along route METRO201 with a mean error of -0.3 dB and a mean squared error of 4.8 dB.

After implementing RDM completely on the GPU, future work will include mapping of other wave guiding effects like reflection or diffraction on building edges onto the graphics hardware. The final aim is a three dimensional radio wave propagation algorithm implemented on the GPU, solely.

ACKNOWLEDGMENT

This work was partially supported by the German Federal Ministry of Economics and Technology within the AIF program PRO INNO II, grant KF0102601SS5.

REFERENCES

[1] E. Damosso, Ed., *COST Action 231: Digital mobile radio towards future generation systems, Final Report*. Luxembourg: Office for Official Publications of the European Communities, 1999.

[2] N. Geng and W. Wiesbeck, *Planungsmethoden für die Mobilkommunikation*. Berlin: Springer, 1998.

[3] Telecommunication Network Consulting GmbH, Aachen. [Online]. Available: http://www.telnetcon.com

[4] M. Schmeink, "Optimierungsalgorithmen zur automatisierten Funknetzplanung." Ph.D. dissertation, RWTH Aachen University, 2005.

[5] P. Wertz, R. Wahl, G. Wölfle, P. Wildbolz, and F. Landstorfer., "Dominant path prediction model for indoor scenarios," in *German Microwave Conference (GeMiC)*, Ulm, April 2005, pp. 176–179.

[6] R. Wahl, G. Wölfle, P. Wertz, P. Wildbolz, and F. Landstorfer., "Dominant path prediction model for urban scenarios," in *14th IST Mobile and Wireless Communications Summit*, Dresden, June 2005.

[7] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Eurographics 2005, State of the Art Reports*, Aug. 2005, pp. 21–51.

[8] N. A. Carr, J. Hoberock, K. Crane, and J. C. Hart, "Fast GPU ray tracing of dynamic meshes using geometry images," in *Proceedings of Graphics Interface*, Quebec, June 2006, pp. 203–209.

[9] D. Weiskopf, T. Schafhitzel, and T. Ertl, "GPU-based nonlinear ray tracing," in *Computer graphics forum*, vol. 23, September 2004, pp. 625–633.

[10] AWE Communications GmbH, Stuttgart, "WinProp." [Online]. Available: "http://www.awe-communications.com"

[11] M. McGuire, *GPU Gems*. Addison Welsey, 2004, ch. Effective Shadow Volume Rendering, pp. 137–166.

[12] G. Mannesmann Mobilfunk GmbH, "Cost 231 - urban micro cell measurements and building data." [Online]. Available: "http://www.ihe.uni-karlsruhe.de/forschung/cost231/cost231.en.html"

[13] A. M. Andrew, "Another efficient algorithm for convex hulls in two dimensions." *Inf. Process. Lett.*, vol. 9, no. 5, pp. 216–219, 1979.