Fast Edge-Diffraction-Based Radio Wave Propagation Model for Graphics Hardware

Tobias Rick, Rudolf Mathar

Institute of Theoretical Information Technology
RWTH Aachen University
D-52056 Aachen, Germany
Email: {rick,mathar}@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. We present a ray-optical approach for wave diffraction at building edges into street canyons, exploiting the programming model of graphics cards. More than twenty predictions per second are achieved in a 7 km^2 urban area with a mean squared error of less than 7 dB when compared with measurements.**

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 evaluated 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 field strength prediction 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 ray-optical prediction algorithms may be found in [5], [6] and [7].

Modern graphics cards offer tremendous computing power due to their highly parallel architecture. Additionally, the performance of graphics cards doubles every half year [8], compared to the performance of standard CPU's which increases with the factor $\sqrt{2}$ every year, according to Moore's Law. Thus, graphics cards form 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 [8]. Recent work includes the mapping of classical ray tracing programs to the GPU, [9] and [10], which is of particular interest with regard to ray-optical wave propagation algorithms.

In this paper, we further develop the approach [11] to use graphics hardware for accelerating field strength predictions. The algorithm developed in Section IV combines the advantage of ray tracing and empirical models in considering building data while still maintaining extremely short running times. Our new algorithm adapts extremely fast shadow algorithms to compute ray paths subject to edge diffraction and the number of obstacles in the direct path.

This paper is organized as follows. In Section II we briefly introduce the underlying programming paradigm of today's graphics cards. Section III presents a path loss model for urban scenarios that derives the received power based on distance, building penetration and diffraction intro street canyons. Section IV describes an algorithm that calculates the corresponding diffraction ray paths. This algorithm is explicitly designed to benefit from graphics hardware. In Section V, a strategy is introduced to optimally calibrate parameters by field measurements. Section VI discusses the introduced algorithm with respect to prediction quality and computation time. We conclude this work with a short summary in Section VII.

II. Graphics Hardware

The underlying architecture of graphics cards is called *Single Instruction Multiple Data* (SIMD), i.e., many parallel processors simultaneously 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

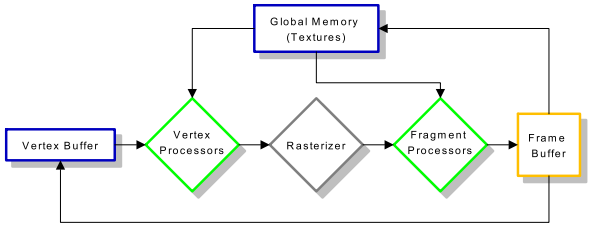


Fig. 1. The Graphics Rendering Pipeline.

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.

For a more detailed introduction into the programming paradigm of today's graphics cards see [8].

III. Path Loss Model

In this section we introduce a path loss model for urban environments. We assume that rays propagate in a straight line from the transmitter and may be diffracted at building edges. The path loss from a transmitter to a receiver point r is modeled by

$$P^{\text{dB}}(r) = c_f + \begin{cases} P^{\text{dB}}_{\text{LOS}}(r), & r \text{ in line-of-sight} \\ P^{\text{dB}}_{\text{LOS}}(r) + P^{\text{dB}}_{\text{ED}}(r), & r \text{ in edge diffraction} \\ P^{\text{dB}}_{\text{NLOS}}(r), & \text{otherwise} \end{cases} \tag{1}$$

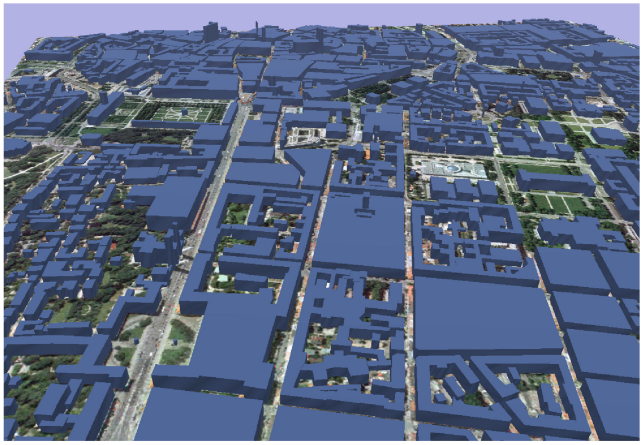


Fig. 2. Building data of Munich, Germany with underlying satellite image.

with

$$P^{\text{dB}}_{\text{LOS}}(r) = \gamma_{\text{LOS}} \cdot 10 \cdot \log_{10}(d_r) \tag{2}$$

$$P^{\text{dB}}_{\text{ED}}(r) = \sum_{i=1}^{K_r} g\left(\alpha_r^{(i)}\right) \tag{3}$$

$$P^{\text{dB}}_{\text{NLOS}}(r) = \gamma_{\text{NLOS}} \cdot 10 \cdot \log_{10}(d_r) + W_r \cdot L_w \tag{4}$$

and the well-known frequency f dependent term

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

with speed of light c. d_r denotes the path length between receiver and transmitter, and $\gamma_{\text{LOS}}, \gamma_{\text{NLOS}}$ the corresponding path loss exponents. K_r is the number of edge diffractions and $\alpha_r^{(i)}$ the i-th diffraction angle. The function

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

with parameters $b_0, b_1, b_2 \in \mathbb{R}$ models the attenuation due to a diffraction angle α. W_r is the number of walls in the direct path from the transmitter to the receiver and L_w the loss per obstructing wall.

Adequate values for the parameters are usually obtained from a calibration with measurement data, see Section V.

IV. Edge Diffraction on the GPU

In the work [11] the benefit of using GPUs for radio wave propagation predictions has been demonstrated by the implementation of the COST-Walfisch-Ikegami (COST-WI) model [1]. In this section we will extend these ideas by including edge diffraction, particularly diffraction into street canyons, enabling us to evaluate the path loss formula (1).

We briefly explain the principles of our algorithm. First the line-of-sight region from the transmitter is determined, see Algorithm 1. For all receiver points in non-line-of-sight the number of obstructing walls is stored. The edge diffraction paths are then calculated by repeated line-of-sight calculation for each diffraction source.

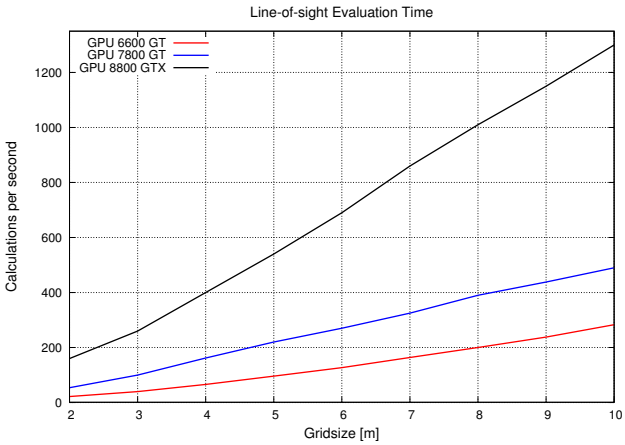


Fig. 3. Line-of-sight calculation time on different generations of graphics cards (Munich scenario 7km^2).

The line-of-sight calculation from [11] is based on shadow algorithms [12] from computer graphics. The basic idea of this technique is to construct a polygonal representation of the shadow cone. The intersection of the shadow cone and the receiver plane is called the *shadow polygon*. Regions are in line-of-sight if they are not inside of any 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).

This method can be implemented efficiently on graphics hardware. The building data, which is usually given as a polygonal ground plot plus one height, see Figure 2, is decomposed into single walls. The vertex processors are responsible for shifting the silhouette points away from the transmitter, 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. This is exactly what procedure *getShadowPolygon* in Algorithm 1 does, which is discussed in detail in [11].

Furthermore, the number of blocking walls at each receiver point can be easily found by slightly modifying these shadow calculations. Instead of a solid drawing (replacement of fragments) of the shadow polygon for each wall, an additive blending (interpolation of fragments) is applied to count the number of shadow fragments at each receiver point.

The construction of edge diffraction paths is described in Algorithm 2. Procedure *getDiffractionSources(NLOSregion)* provides all possible diffraction sources for a given region, i.e., all building edges that are on the border between line-of-sight and non-line-of-sight. The combined line-of-sight regions from each visible building edge results in the region that can be reached by the first level of edge diffraction. When combining overlapping line-of-sight regions only the source point with the lower diffraction angle is stored, because a lower path loss is expected. This can easily be achieved by means of the depth buffer. Applying this algorithm recursively and recording the corresponding source points (i.e. building edges) for the line-of-sight regions leads to the construction of edge diffraction paths of arbitrary depth.

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

```
CLEARBUFFER(colorbuffer,RGB(LOS))
for all wall ∈ W do
  {In the vertex processor}
  poly ← GETSHADOWPOLYGON(wall,T)
  {In the rasterizer}
  fragments ← RASTERIZEPOLYGON(poly)
  {In the fragment processor}
  for all frag ∈ fragments do
    frag.color ← RGB(NLOS)
  end for
end for
return colorbuffer
```

Algorithm 2 CALCEDGEDIFFRACTION(WallSet $\mathcal{W}$, Transmitter T)

```
NLOSregion ← GETNLOSREGION(W,T)
{NLOS region of transmitter}
D_s ← GETDIFFRACTIONSOURCES(NLOSregion)
{Get all possible diffraction sources}
CLEARBUFFER(Framebuffer,RGB(noDiffraction))
CLEARBUFFER(Depthbuffer,π)
for all d ∈ D_s do
  NLOSfromSource ← GETNLOSREGION(W,d)
  {Get LOS region for each diffraction source}
  for all frag ∉ NLOSfromSource do
    frag.color ← RGB(d) {Diffraction source}
    frag.depth ← α {Diffraction angle}
    {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
return Framebuffer
```

The main building block of the presented algorithm is Algorithm 1 *getNLOSRegion*. This is the most frequently used component and hence the term which influences computation time most of all. The implementation of this procedure is therefore adapted from [11], leading to more than 500 line-of-sight calculations per second at a resolution of five meters on a graphics cards like the NVIDIA GeForce 8800 GTX, see Figure 3. The reason for this tremendous amount of line-of-sight computations per second is the well embedded implementation on graphics processing units. Additionally, the implementation of algorithms on graphics hardware benefits from the enormous speedup between subsequent generations of graphics cards.

Prediction model	mean error	MSE	std. dev.	Runtime
Ericsson (Raytracing + COST-WI), [1]	2.3 dB	-	7.1 dB	-
Uni.-Karlsruhe (Raytracing), [1]	2.4 dB	-	9.1 dB	-
CORLA [13]	0.1 dB	4.2 dB	4.2 dB	8 s
COST-WI (GPU) [11]	10.7 dB	12.6 dB	6.6 dB	0.0045 s
RDM (GPU) [11]	−0.2 dB	4.7 dB	4.7 dB	3.05 s
Model from Section III (GPU)	**0.1 dB**	**4.5 dB**	**4.5 dB**	**0.05 s**

TABLE I

ACCURACY OF PROPAGATION MODELS IN THE COST-MUNICH SCENARIO ALONG ROUTE METRO201.

V. PARAMETER CALIBRATION

Influencing factors that are usually not included in the urban model like building material, roof style, texture, vegetation, etc. are covered by an adjustable parameter vector, see also [13]. A city with modern skyscrapers, mainly with glass fronts and flat roofs, and a small town, mainly with pitched roofs and stone fronts, will certainly be subject to different attenuation patterns.

In introduction to multivariate analysis and least squares estimation used for calibration is given in [14]. All parameters introduced in Section III are merged into a vector $\mathbf{z} = (\gamma_{LOS}, \gamma_{NLOS}, b_0, b_1, b_2)$. $\mathcal{Z}$ denotes the set of all feasible parameter vectors. Further, attenuation measurements $M^{\text{dB}}(r)$ are given for receiver points $r \in \mathcal{R}$, with $N = |\mathcal{R}|$. The dependency of the path loss formula (1) on the vector $\mathbf{z}$ is indicated by writing $P^{\text{dB}}(r, \mathbf{z})$. A common performance index is the *mean squared error* (MSE)

$$\sqrt{\left(\frac{1}{N}\sum_{r\in\mathcal{R}}\left(M^{\text{dB}}(r) - P^{\text{dB}}(r,\mathbf{z})\right)^2\right)} \tag{5}$$

which is minimized by

$$\mathbf{z}^* := \arg\min_{\mathbf{z}\in\mathcal{Z}} \sum_{r\in\mathcal{R}} \left(M^{\text{dB}}(r) - P^{\text{dB}}(r,\mathbf{z})\right)^2. \tag{6}$$

By solving (6), path loss predictions are easily adjusted to different types of environment.

VI. RESULTS

For benchmarking purposes we used the building and measurement data released in the COST 231 action for the city of Munich, Germany [15]. In the following, all path loss calculations are based on the parameter vector obtained by the calibration on the measurement points along route METRO202 in the COST 231 Munich scenario. All predictions were performed for the whole supplied area of approximately 7 km^2 with a resolution of 5 m. Figure 4 visualizes a field strength prediction with the above introduced wave propagation algorithm. We observe, that in the particular Munich scenario the first level of edge diffraction already provides sufficient prediction accuracy when used in combination with (4). This is mainly because (4) approximates roof diffraction effects in the non-line-of-sight region further away from the transmitter.

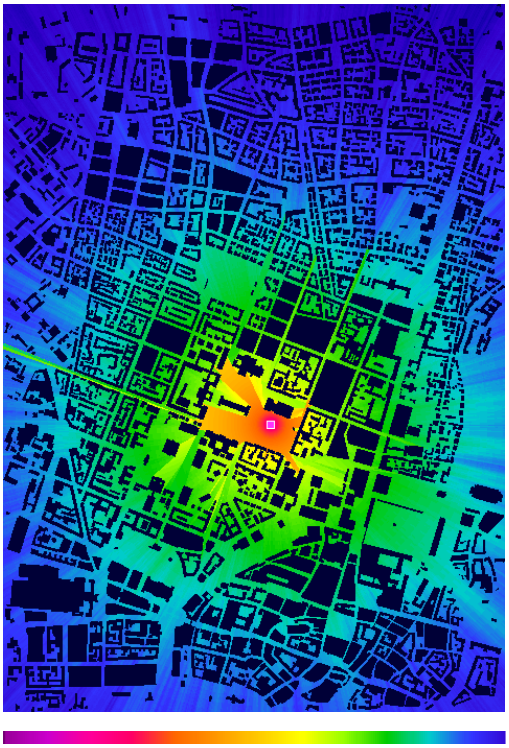


Fig. 4. Field strength prediction in Munich.

Measurement Route	mean error	MSE	std. dev.
METRO200	2.1 dB	6.2 dB	5.9 dB
METRO201	0.1 dB	4.5 dB	4.5 dB
METRO202	-0.1 dB	5.7 dB	5.7 dB

TABLE II

ACCURACY OF THE PRESENTED MODEL IN THE COST-MUNICH SCENARIO ALONG DIFFERENT MEASUREMENT ROUTES.

Table I compares the accuracy and runtime of the presented field strength prediction algorithm and previously published results in [1], [11] and [13]. It can be seen that our algorithm reaches the prediction quality of highly tuned ray launching tools like CORLA [13] with an extraordinary fast computation time of under 0.05 s. Hence, with our implementation more than 20 field strength predictions are possible in less than one second. This allows real-time computation of wave propagation, greatly improving the overall runtime of network optimization algorithms. Furthermore, such a reduction of runtime completely eliminates the need for pre-computations of field strength predictions.

VII. Conclusions

The parallel architecture of today's graphics cards is exploited to achieve extremely fast computation times. With the first level of edge diffraction, computation times are as fast as 0.05 seconds on a recent graphics card (NVIDIA 8800 GTX) and high quality predictions with a mean squared error between 4.5 dB and 6.2 dB are achieved in the COST-Munich scenario (7 km^2), see Table II and Figure 5.

Future work will include mapping of other wave guiding effects like reflection or diffuse scattering onto graphics hardware. The final aim is a three dimensional radio wave propagation algorithm solely implemented on the GPU.

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. Thesis, 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] L. M. Correia, Ed., *COST Action 273: Mobile Broadband Multimedia Networks, Final Report*. Academic Press, 2006.

[8] 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.

[9] 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.

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

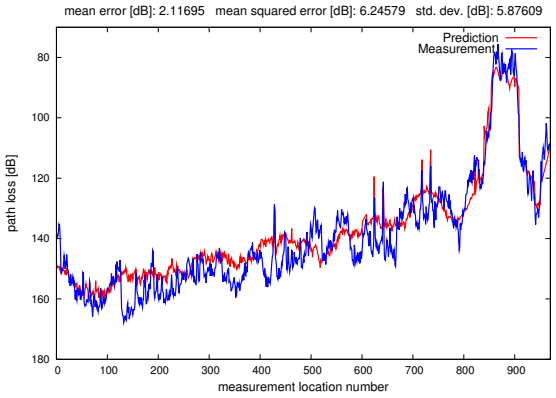
[11] D. Catrein, M. Reyer, and T. Rick, "Accelerating radio wave propagation predictions by implementation on graphics hardware," in *To appear in: Proceedings: IEEE VTC Spring*, 2007.

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

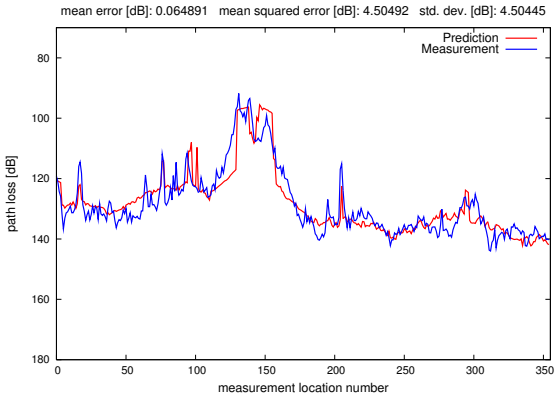
[13] R. Mathar, M. Reyer, and M. Schmeink, "3d ray launching algorithm for urban field strength prediction," Theoretische Informationstechnik, RWTH Aachen, Tech. Rep., 2006.

[14] K. V. Mardia, J. T. Kent, and J. M. Bibby, *Multivariate Analysis*. London: Academic Press, 1979.

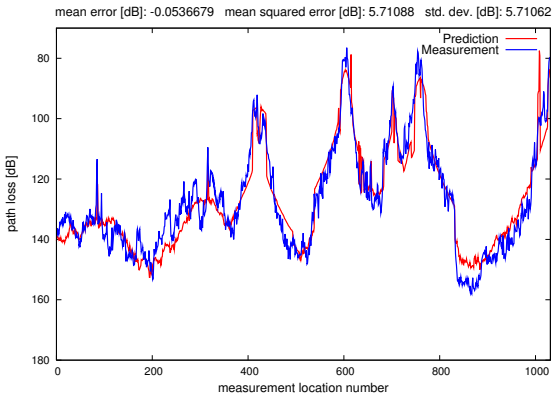
[15] G. Mannesmann Mobilfunk GmbH, "Cost 231 - urban micro cell measurements and building data." [Online]. Available: "http://www.ihe.uni-karlsruhe.de/forschung/cost231/cost231.en.html"



(a) Route METRO200



(b) Route METRO201



(c) Route METRO202

Fig. 5. Comparison between measured and predicted path loss in the COST 231 Munich scenario.