Accelerating Radio Wave Propagation Algorithms by Implementation on Graphics Hardware

Tobias Rick and Torsten Kuhlen
RWTH Aachen University, JARA-SIM
Germany

1. Introduction

Radio wave propagation prediction is a fundamental prerequisite for planning, analysis and optimization of radio networks. For instance coverage analysis, interference estimation or channel and power allocation all rely on propagation predictions. In wireless communication networks optimal antenna sites are determined by either conducting a series of expensive propagation measurements or by estimating field strengths numerically. In order to cope with the vast amount of different configurations to select the best candidate from and to avoid expensive measurement campaigns, numerical predictions have to be both accurate and fast. In this chapter we focus on accelerating techniques for radio wave propagation algorithms in dense urban environments with the target frequency range of common mobile communication systems, i.e., several hundred MHz up to few GHz. One important aspect in radio wave propagation is the prediction of the mean received signal strength which can be simulated by taking complex interactions between radio waves and the propagation environment (see Figure 1) into account. Thus, the simulation of radio waves for propagation predictions becomes a computationally intensive task.
A promising approach is the use of ordinary graphics cards, nowadays available in every personal computer. With over 1000 Gigaflops, modern graphics hardware offers the computational power of a small-sized supercomputer. This is achieved by a strict parallel many-core architecture which can be accessed by a high level of programmability. The main challenge of utilizing graphics hardware for scientific computations is to trick the graphics processors into general purpose computing by casting problems as graphics: Input data is transformed into images and algorithms are turned into image synthesis. However, in the last couple of years a growing support of so-called ”General Purpose Computation on Graphics Hardware” has led to recent changes in this architecture, allowing more common ways of parallel programming. Much effort and interest has been put on the acceleration of ray optical approaches, since most ray tracing algorithms tend to be computational intensive and exhibit run times up to hours. Therefore, we focus on the efficient implementation of wave guiding effects on graphics hardware. Among the most time consuming tasks in ray tracing is the problem of visibility between objects, i.e., the identification of all possible interaction sources for diffracted or reflected propagation rays. The algorithms we will present here are specifically designed to reduce the computational cost of the visibility computations by exploiting special features of the graphics card.

Source: Wave Propagation in Materials for Modern Applications, Book edited by: Andrey Petrin,
ISBN 978-953-7619-65-7, pp. 526, January 2010, INTECH, Croatia, downloaded from SCIYO.COM

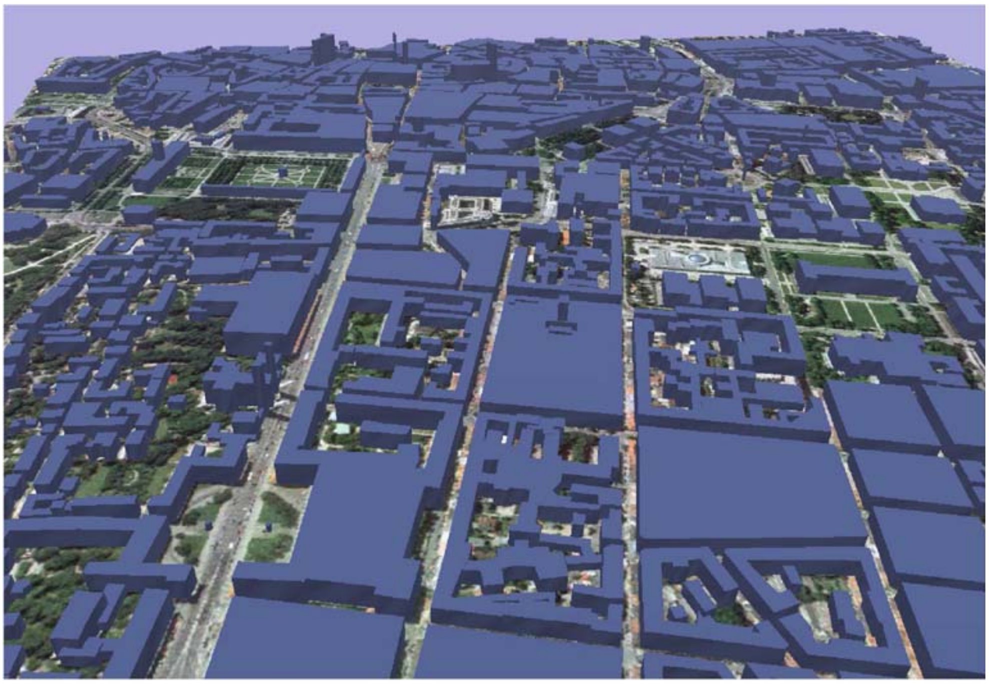


Fig. 1. Typical propagation environment with 2.5 dimensional building structures. The underlying satellite image provides geographical reference.

In principle our method is based on ray tracing. However, interaction sources are found by tracing full line-of-sight beams instead of single rays. Each beam is constructed such that it covers all rays that fall inside its angular opening. Thus, fewer beams than rays have to be processed. Furthermore, we do not actually compute any intersections between propagation environment and radiation source. In fact, we use a discrete sampling approach, i.e., ray tracing is accomplished by traversing discrete points of all rays that lie within a certain beam. Our approach results in an acceleration of wave propagation algorithms by developing a special implementation that exploit the parallel architecture of modern graphics hardware. As prove of concept, we have implemented four basic propagation effects: (1) line-of-sight propagation, (2) wall penetration depth in non-line-of-sight, (3) diffraction into street canyons and (4) diffraction over building rooftops.
The remainder of this chapter is organized as follows. After briefly reviewing previous work in Section 2, we give an overview of the general concept of programmable graphics hardware in Section 3. Then, we cover the basic prerequisites of our algorithm in Section 4 and present our method for the computation of urban propagation phenomena in Section 5. Our path loss model is described in Section 6. We finally give a performance analysis in terms of computation time and accuracy in Section 7 and conclude the chapter in Section 8.

2. Related work

A theoretical foundation of radio wave propagation is given for instance by Rappaport (1995). Since a variety of approaches exists for solving the problem of predicting mean received signal strength we point the reader to Correia (2006) for an overview. In general, we distinguish between empirical (stochastic) channel models and deterministic propagation algorithms. Empirical models, approximate the actual path loss by parametrized functions which are commonly accompanied by extensive measurement campaigns, whereas deterministic approaches are often based on the principle of ray tracing. That is, they identify ray paths through the propagation environment based on wave guiding effects like reflection or diffraction.

Well-known examples of empirical models are the work of Hata (1980) and Ikegami et al. (1984). They propose to model the radio propagation phenomena by approximating the actual propagation loss (path loss) by parametrized functions. Hata determined parameter values by conducting extensive measurement campaigns. Ikegami extended Hata's work by analyzing the dependence of approximate equations of mean field strength in urban propagation environments with respect to height gain, dependence on street width, propagation distance and radio frequency. More recently, Erceg et al. (1999) presented a stochastic channel model which can be applied to frequency ranges above two GHz (WiMAX). Additionally to height gains and propagation distance, Erceg included a parametrization of the environment into his model according to a flat or hilly terrain with either high or low vegetation. Empirical models are typically characterized by short evaluation time but are prone to huge prediction errors and perform especially poor in heterogeneous propagation environments like historically grown cities, cf. Damosso (1999).

Therefore, most deterministic algorithms rely on the computation of actual propagation paths due to wave guiding effects like reflection, diffraction and scattering. Typical approaches are often based on ray tracing which was originally introduced by Whitted (1980) to compute global illumination effects based on geometric optics for image synthesis. Although, global illumination as formulated by Kajiya (1986) and radio wave propagation are similar problem statements, different propagation effects like diffraction or interference become dominant when shifting from visible light to radio waves due to the different size of wavelengths. There are various approaches that focus on acceleration techniques by mapping global illumination algorithms onto the GPU, among them are publications by Horn et al. (2007); Carr et al. (2006) and Dachsbacher et al. (2007). Global illumination techniques have been used for different problems before, for instance for sound rendering. Notable here are the works of Tsingos et al. (2001; 2004) and Funkhouser et al. (1999). Some work on the diffraction effect is described for instance by Stam (1999) for application in computer graphics, whereas Tsingos et al. apply diffraction theory for modeling acoustics in virtual environments.

Ikegami et al. (1991) showed that classical ray tracing can also be applied to the estimation of radio propagation losses. Due to complex interaction of radio waves and geometric structures, this is a time consuming task. However, high prediction accuracy can be achieved. For instance, Schaubach et al. (1992); Erceg et al. (1997); Kim et al. (1999) and Schmitz & Kobbelt (2006) state that their predicted path loss values were generally within 4 to 8 dB of the measured path loss which is considered as a very good result. In Mathar et al. (2007) a ray launching algorithm is used which represents an urban environment as a grid of discrete blocks. Ray-object intersections are found by traversing the blocks by a line sampling method, thereby greatly reducing computation time. In order to further reduce the computational complexity Rick et al. presented an GPU-based approach to radio wave propagation in Catrein et al. (2007) and Rick & Mathar (2007). They trace propagation paths in a discrete fashion by repeated rasterization of line-of-sight regions. Propagation predictions are computed at

interactive rates. However, the accuracy depends on the resolution of the rasterization. Part of their work is presented here. A more general overview of general purpose computations on GPUs and on the evolution of GPU architectures is given for instance by Owens et al. (2005) and Owens et al. (2008), respectively.
The idea of ray tracing can be further extended to the concept of beams, which are a continuum of rays. Beam tracing was first introduced by Heckbert & Hanrahan (1984). The main benefits of beam tracing can be summarized as reduced intersection tests and less sampling problems since after a few iterations ray samples tend to become either too sparse or too dense. Published work in this area includes application of real time rendering by Overback et al. (2007) or audio rendering by Funkhouser et al. (1999). An application of beam tracing to the problem of radio wave propagation can be found in the work by Rajkumar et al. (1996) and more recently by Schmitz et al. (2009). They especially address the issue of delay spread due to multi path propagation.

3. Graphics hardware

Up to 1999 graphics cards had a non-programmable so-called *fixed-function* architecture. Over the last decade they evolved to configurable pipelines and recently into fully programmable floating-point *graphics processing units* (GPUs).
Modern GPUs are extremely powerful computing devices. Figure 2 depicts the evolution of *floating point operations per second* (FLOPS) of the GPU in comparison to the CPU over the last few years, c.f. Owens et al. (2005). The performance gain of graphics cards roughly doubles every half year, clearly outperforming the CPU when competing for Giga FLOPS (GFLOPS). The latest NVIDIA G80 GPU achieves over 300 GFLOPS. The performance of a Quad-Core Intel Xeon processor (3GHz) is roughly 80 GFLOPS Intel Corporation (2008).
The GPU is specialized for computational intensive, highly parallel calculations. Rather than caring for data caching and flow control as the CPU, the GPU is especially designed to support data processing. The GPU architecture follows the *Single Instruction Multiple Data* paradigm. Many processors simultaneously execute the same instructions on different parts of a datastream. The key challenge of programming graphics hardware is to correctly map problems to the graphical rendering context. Input data is transformed into images or geometry and algorithms are turned into image synthesis.

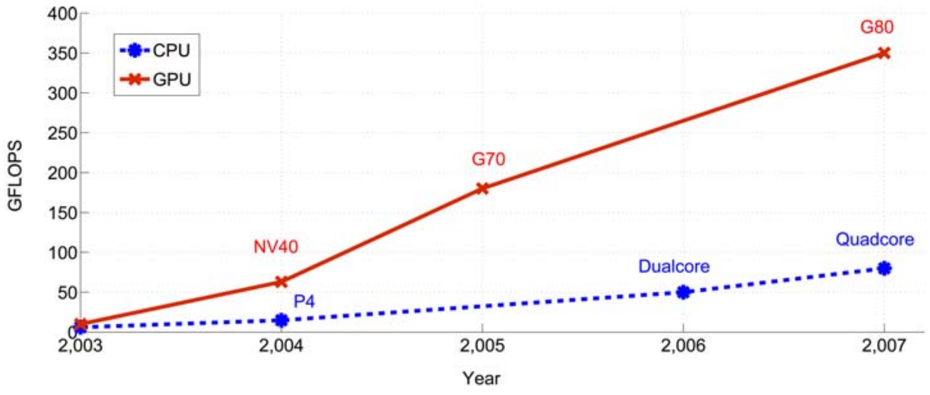


Fig. 2. Floating point operations per second: GPU vs. CPU. The graphics card clearly outperforms the CPU in terms of floating point operations.

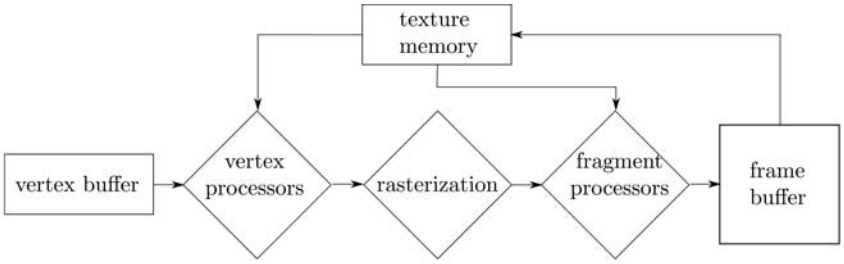


Fig. 3. The graphics rendering pipeline. Geometric primitives are loaded into the vertex buffer and transformed by multiple vertex processors in parallel. The rasterization samples the geometry into pixel positions such that multiple fragment processors can assign colors that are recorded in the frame buffer for the final image.

In general, a computation on the GPU consists of a pass through the various stages of the so-called *graphics rendering pipeline*, see Figure 3. We refer to this pass as a *rendering pass*.
Basic geometrical primitives like points or triangles are loaded into the *vertex buffer*. The primitives are described by their location in space, i.e., coordinates (*vertices*) and associated attributes like material or color. Additionally, data arrays (*textures*) of integer or floating point values can be allocated directly in graphics memory.
First, multiple vertex processors execute in parallel the instructions from a user-written program (*vertex program*). Vertex programs operate on single vertices with access to their attributes and global read-only texture memory. Typically, geometric transformations like translation, rotation and projection are applied.
In the subsequent step, the transformed geometry is sampled (*rasterized*) into discrete points (*fragments*). Each fragment corresponds to a single *pixel* (picture element) position on the screen, and typically has additional information like color and a depth value, i.e., the distance between viewer and the object which originally corresponded to the pixel.
The fragment processors operate analogous to the vertex processors. A user-written *fragment program* is executed on each fragment in a parallel fashion. The fragment program is executed *once per fragment*. Most instructions are floating-point vector operations. Typically, lighting models are evaluated and corresponding fragment colors are assigned (*shaded*).
The *frame buffer* is a two-dimensional array of pixels. The task of the frame buffer is to assemble the final result of the GPU computation by collecting and recording all fragments. *Frame buffer operations* decide how the color from the incoming fragments is combined with the color already stored at the same pixel position. Thus, many fragments can contribute to the final color of a pixel. Commonly, the frame buffer contains color information and hence, it can be displayed on the screen. Information in the frame buffer can also be read back to main memory, which is especially useful for GPU computations. For further readings we point the reader to introductory texts and advanced techniques on graphics and shader rogramming from Akenine-Möller & Haines (2002); Shreiner et al. (2003) or Fernando & Kilgard (2003).

4. Overview

Similar to Wahl et al. (2005), the data requirements for our algorithms are building structures with corresponding building heights. The shape of rooftops is usually omitted

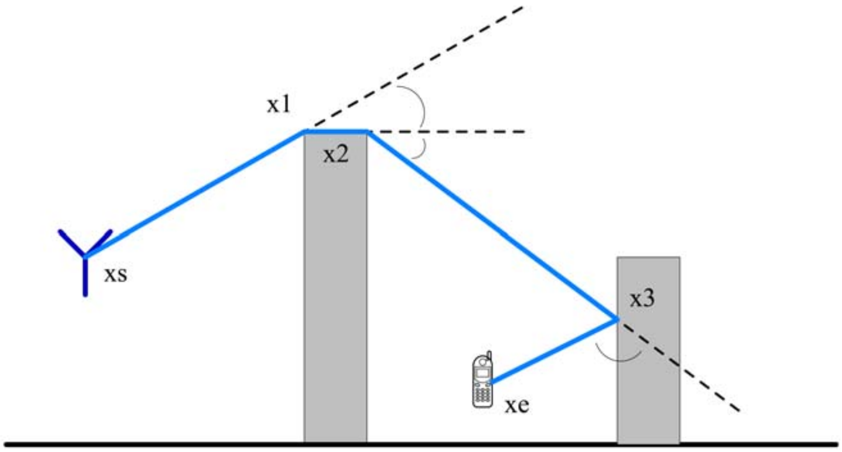


Fig. 4. Example propagation path (side profile). A ray is emanating from the radiation source, diffracted at the rooftop of the first building and reflected the second building into the street: $x_s \rightsquigarrow x_1 \rightsquigarrow x_2 \rightsquigarrow x_3 \rightsquigarrow x_e$.

and hence, a building is described by its polygonal outline and one height value. We refer to this representation as 2.5 dimensional. Building structures ares usually given in vector format with a location accuracy in the order of 1 to 10 meter. In order to produce reliable results, height accuracy should be around 1 to 2 meter. Information about vegetation is not considered and terrain is assumed to be flat. Figure 1 shows a typical propagation environment.

The basic propagation phenomena are reflection, diffraction and scattering. All effects contribute to the radio signal distortions and give rise to signal fluctuations (fading) and additional signal propagation losses. We distinguish propagation effects according to the characteristics of the propagation environment by approximating which propagation paths (see Figure 4) are most likely to occur. In the course of this chapter we will present specialized algorithms for each of the following propagation effects.

If receiver and transmitter antenna have an unobstructed direct path, they are in *line-of-sight* (LOS), otherwise in *non-line-of-sight* (NLOS). An urban *high-rise* scenario consists predominantly of streets lined with tall buildings of several floors each. High building heights make significant contributions from diffraction over multiple rooftops rather unlikely. Therefore, if transmitter antennas are mounted below rooftops, dominant propagation effects are expected from reflection and diffraction into street canyons. See Figure 8(b) for an illustration, more details can be found in Andersen et al. (1995). Furthermore, an urban *low-rise* scenario is characterized by wide streets and buildings with less than three floors. Antennas are usually mounted above average rooftop level and diffraction over rooftops (Figure 9(a)) becomes the dominant propagation effect. Propagation paths due to reflection are not considered in this chapter.

Hence, we formulate the requirements of our propagation algorithm as follows: (1) It is necessary to efficiently distinguish between regions in LOS and in NLOS. (2) In case of NLOS, we require our algorithm to compute different propagations paths, namely wall penetration depth, diffraction into street canyons and diffraction over rooftops.

5. Algorithm

The input for our algorithms is a database of building structures as described in Section 4. In general, the output is a discrete two-dimensional raster image which serves as a role model for the discrete representation of results from GPU computations. An image consists of pixels that are organized in a regular array. Pixels are the data elements of this structure. However, pixels are not restricted to contain only color information. Pixel data is interpreted according to the current context of the GPU computation.

In order to provide a formal description of the presented algorithms we introduce the following notation (see Table 1): Let $s = (x,y, z)$ be a radiation source location, e.g., the transmitter antenna, where the height is referred to as $s.z$. All height values are relative to ground. A wall w of a building structure consists of two points p_0 and p_1 at ground level and two points p_2 and p_3 at rooftop level. The set containing all walls w is denoted by $\mathcal{W}$. Additionally we associate a normal vector $\vec{n}$ with each wall. Normal vectors are perpendicular to their wall and point away from the corresponding building.

Since the terrain is assumed to be rather flat and the receiver points are typically located 1.5 meter above ground we define a *receiver plane* $\mathcal{R}$ to be a discrete set of receiver points at constant height $\mathcal{R}.z$. We restrict ray path calculations to only those paths that intersects the receiver plane.

In the following we will describe algorithms which are explicitly designed to run directly on graphics hardware. First, we present a method for determining line-of-sight regions. Then we will show how this algorithm can be extended to provide additional non-line-of-sight information. Furthermore, we show how the graphics card can be exploited for ray path calculation due to diffraction into street canyons and diffraction over rooftops. GPU implementations are obtained by separating the calculation of these effects into distinct algorithms.

5.1 Line-of-sight

The *line-of-sight* (LOS) algorithm computes a sampling of the receiver plane $\mathcal{R}$ where each point $p \in \mathcal{R}$ is marked whether it is in clear line-of-sight to a source s or not. This algorithm will be one of the main building blocks in the computation of ray paths on graphics hardware. The GPU computation is based on the concept of so-called *shadow volumes*, c.f. Fernando (2004). This technique constructs a polygonal representation of the shadow cone (shadow volume) for 3-dimensional triangular geometry. Since the description of urban propagation environments involves usually just a polygonal outline and one corresponding height value, we propose a specialized algorithm for this particular form of geometry (see Catrein et al. (2007)).

$s = (x,y,z)$	Radiation source location with height $s.z$
$\mathcal{R}$	Discrete set of receiver points with constant ground level $\mathcal{R}.z$
$\mathcal{W}$	Set of all building walls w
$w = (p_0, p_1, p_2, p_3, \vec{n})$	Building wall with p_0, p_1 at ground level and p_2, p_3 at rooftop level. The normal vector $\vec{n}$ is perpendicular to the wall and points away from the associated building.
$\mathrm{NLOS}(s \rightarrow w)$	Shadow polygon cast by wall w when viewed from point s
$\mathrm{LOS}(s) \subseteq \mathcal{R}$	Subset of $\mathcal{R}$ that is in line-of-sight to s

Table 1. Notation

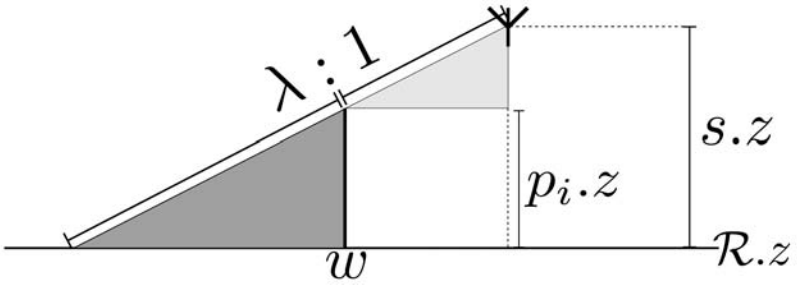


Fig. 5. Size of the shadow polygon (dark gray) according to the intercept theorem for a source height $s.z$ greater than the wall height $p_i.z$ (side profile).

The main idea is to extract the shadow of each building wall directly in the receiver plane. We refer to the intersection of the shadow cone and the receiver plane as *shadow polygon*. Regions are in LOS, if and only if they are in no shadow polygon.

The construction of the shadow polygon of a wall w proceeds as follows: each shadow polygon is a quadrangle with corners (q_0, q_1, q_2, q_3). Points q_0 and q_1 are given by the corners of the wall p_0 and p_1 on the ground. The remaining two corners are determined by the intersection of the receiver plane $\mathcal{R}$ and each of the straight lines through the source point s and the wall point at roof level p_2 and p_3.

According to the intercept theorem (see Figure 5), q_i, $i \in \{2, 3\}$ is then given by

$$q_i = p_i + \lambda \cdot (p_i - s) \tag{1}$$

where

$$\lambda = \begin{cases} \frac{p_i.z - \mathcal{R}.z}{s.z - p_i.z} & ,\text{if } s.z - p_i.z > 0 \\ \infty & ,\text{otherwise} \end{cases} . \tag{2}$$

The corners of each shadow polygon are computed by a vertex program in parallel for each wall and sampled into discrete points by the subsequent rasterization phase. The result is a two-dimensional pixel array, every (discrete) receiver location lies either in LOS or inside a shadow polygon, hence in NLOS.

With q_2 and q_3 according to equation (1), we refer to the shadow polygon of a single wall w with source point s as

$$\text{NLOS}(s \rightarrow w) = \{p \in \mathcal{R} \mid p \in \text{polygon}(w.p_0, w.p_1, q_2, q_3)\} \tag{3}$$

A *LOS beam* LOS(s) is the set of discrete points in the receiver plane that do not lie in any shadow polygon of the source s

$$\text{LOS}(s) = \bigcap_{w \in \mathcal{W}} \{p \in \mathcal{R} \mid p \notin \text{NLOS}(s \rightarrow w)\} . \tag{4}$$

Figure 6 shows an example of a discretized receiver plane with pixels in LOS (dark grey) and NLOS (light grey) pixels. Note, that although the result is two-dimensional, all shadow computations are performed using all 2.5-dimensional information.

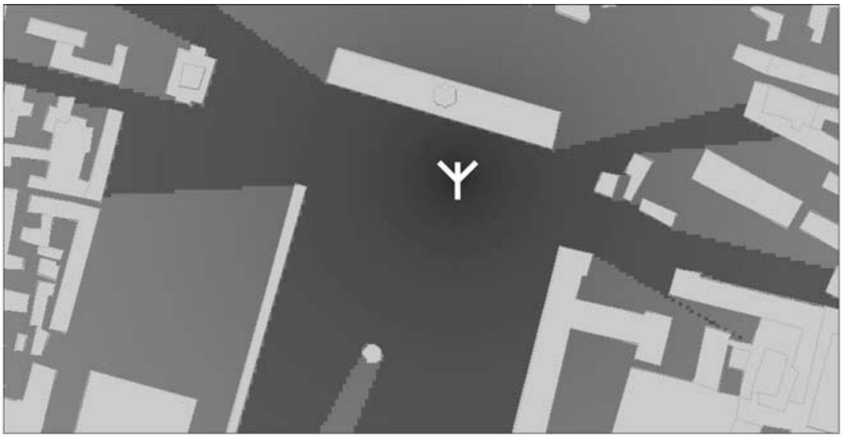


Fig. 6. Top view of the discrete line-of-sight region (dark grey) with the source point in the center (white cross). Building interior is shown in light grey.

5.2 Wall transmission

The algorithm for LOS beams can be extended to provide additional *non*-line-of-sight information. This can include for instance the number of penetrated walls or material which can be achieved by taking all walls into account that intersect the direct ray from the source to a receiver as sketched in Figure 7.

We first provide a more thorough look at GPU frame buffer operations which are an integral part of the algorithm for transmission depth. When fragments are collected and recorded in the frame buffer at the final stage of the rendering pipeline (c.f. Section 3), frame buffer operations decide how fragments that fall on the same pixel position, contribute to the final color of that pixel. Commonly, the fragment with the lowest depth value, i.e., which is nearest to the viewer, determines (replaces) the pixel color. Alternatively, the final color can be a combination (interpolation) of the values of both fragments, the one already in the frame buffer and the new one, which wants to occupy the same pixel position. This technique is called *blending*. In image synthesis, blending is commonly used to draw translucent objects.

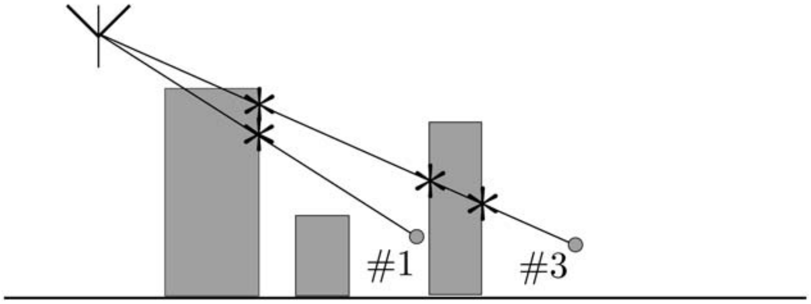


Fig. 7. Wall transmission depth can be taken into account by counting the number of walls in the direct path between the source and the receiver points. On the graphics card, this is achieved by an additive blending of shadow polygons.

Here, we use the blending capability to increase the value in the frame buffer with every rendered shadow polygon. Thus, instead of a solid rendering (replacement of fragments) of the shadow polygons, we apply an additive blending. This effectively counts the number of shadow fragments at each receiver location.

However, frame buffer operations like blending are currently implemented in hardware with a precision of 8 bits. This would means that we could only count up to 255 wall transmissions, which may not be enough for large and complex scenarios. A solution is presented by graphics cards that offer buffers (textures) of higher precision like 32 bit. Blending has to be implemented by a user-written fragment program since high precision blending is not yet directly supported in hardware.

To overcome this drawback and to support propagation environments with an arbitrary number of walls we propose the following hybrid approach between 8 bit hardware and 32 bit fragment shader blending.

Two 32 bit buffers are created and filled with zeros. These buffers will provide a so-called ping-pong scheme because current GPUs do not support a simultaneous read and write access to the same buffer. One buffer is referred to as the *next* buffer, it is the buffer which will be rendered to. The other buffer is referred to as the *current* buffer, it is bound as a texture and read from in the updating step. After each pass, the buffers are swapped, so that the *next* buffer becomes the *current* buffer and vice versa.

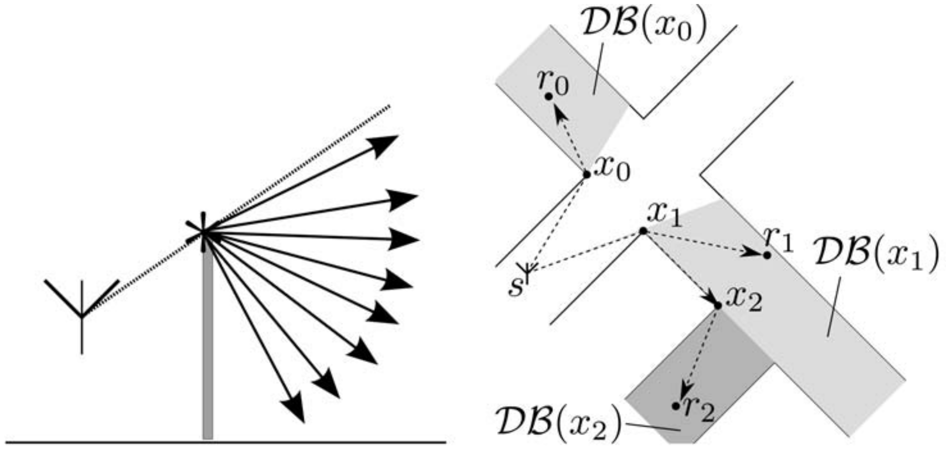
Hence, one buffer will keep track of the total transmission depth for every fragment and the other will store intermediate results. The actual rendering of the shadow polygons is done into a third 8 bit frame buffer with hardware supported additive blending. The vertex buffer containing the wall geometry is rendered in separate chunks of 255 walls each. After each chunk rendering, the transmission depth of every fragment is added to *next* buffer. This is done by a fragment shader which reads the *current* buffer and the 8 bit buffer containing the transmission depths of the latest wall chunk. A simple add operation is performed and the result is stored in the *next* buffer. This procedure is repeated until all wall chunks have been processed.

This approach is a multi-pass algorithm, for n walls it requires $\lceil \frac{n}{255} \rceil$ rendering passes. This number can be reduced, if chunks are rendered in parallel whose shadow polygons do not overlap. Roughly the same performance as the original LOS algorithm is achieved since this approach utilizes parts of the rendering pipeline (frame buffer operations) that have been idle before.

5.3 Diffraction into street canyons

Classical ray tracing algorithms commonly model diffraction effects by tracing a multitude of rays into the respective diffraction cone as illustrated in Figure 8(a). This is computationally quite intensive, as one ray (the one hitting the diffraction edge) is split into many secondary rays. Shooting fewer rays, usually results in an under sampling of the propagation environment, leading to regions where no rays arrive. Thus, no path loss calculations can be performed there.

A solution to this problem is presented by the following approach. In order to achieve a high throughput of diffraction computations, we trace full beams instead of single rays. This has the advantage that beams cover a lot more area than single rays and thus, a lot fewer beams than rays have to be processed for a sufficient sampling of the propagation environment.



(a) (b)

Fig. 8. (a) Modeling diffraction by shooting a multitude of rays into the diffraction cone. (b) Diffraction beams in nearby street canyons. Diffraction sources are $\mathcal{DS}(s) = \{x_0, x_1\}$ and $\mathcal{DS}(x_1) = \{x_2\}$. Geometric ray paths are for instance $s \rightsquigarrow x_0 \rightsquigarrow r_0$ or $s \rightsquigarrow x_1 \rightsquigarrow x_2 \rightsquigarrow r_2$.

The algorithm consist of three main steps. (1) Identify all potential diffraction sources for a propagation into street canyons. (2) Determine the propagation paths of the secondary wave fronts. (3) Compute geometrical paths in reverse order from the target location towards the diffraction source. This procedure can be applied recursively if propagation paths along multiple street canyons are desired. The idea is sketched in Figure 8(b).
Let s be a radiation source. According to Section 5.1 the set of discrete points that are in line-of-sight to s is defined by LOS(s). We define a *diffraction source* as an end point of a wall w that satisfies the following criteria: (1) The wall is in LOS to the radiation source if one of the end points is in LOS

$$w \in \mathrm{LOS}(s) \Longleftrightarrow w.p_0 \in \mathrm{LOS}(s) \vee w.p_1 \in \mathrm{LOS}(s) \tag{5}$$

and (2) if the normal vector of the wall is facing away from from the radiation source

$$\langle w.\vec{n}, \| w.p_0 - s \|_2 \rangle < 0. \tag{6}$$

For a radiation source s, we collect all possible diffraction sources in the set $\mathcal{DS}(s)$

$$\mathcal{DS}(s) = \bigcup_{w \in \mathrm{LOS}(s)} \begin{cases} \left\{ w.p_j \right\} & ,\text{if } \langle w.\vec{n}, \| w.p_0 - s \|_2 \rangle < 0 \\ \varnothing & ,\text{otherwise} \end{cases} \tag{7}$$

where each $w.p_j$ is choosen such that

$$j \in \{0,1\} \wedge \| w.p_j - s \|_2 \leq \| w.p_{1-j} - s \|_2 \,. \tag{8}$$

Hence, a diffraction source is always a point on the wall that is closest to the radiation source.

Now, we construct the diffraction beam $\mathcal{DB}(x)$ as a secondary wave front that originates from the diffraction source $x \in \mathcal{DS}(s)$. With our notation we can write a diffraction beam as

$$\mathcal{DB}(x) = \{r \in \mathcal{R} \mid r \in \mathrm{LOS}(x) \wedge r \notin \mathrm{LOS}(s)\} \tag{9}$$

All points which have been in line-of-sight to the original radiation source s will not be part of the diffraction beam. Thereby, we ignore regions that would result in very large diffraction angles which would in turn not contribute to the overall signal level, significantly.

The final steps consists of the reconstruction of geometric ray paths based on the beam information. For each beam $b \in \mathcal{DB}(x)$, we create a set of ray paths $\mathcal{RP}(b)$

$$\mathcal{RP}(b) = \{s \rightsquigarrow x \rightsquigarrow r \mid r \in \mathcal{DB}(x)\} \tag{10}$$

Thus, every point within the beam travels along its diffraction source towards the original radiation source. Diffraction paths of arbitrary length can be constructed by assigning the start points of the rays as new diffraction sources in a recursive fashion.

The computational bottleneck of this method are the numerous LOS computations. Therefore, we propose to use the LOS implementation on the GPU as described in Section 5.1.

5.4 Propagation over rooftops

The algorithm for diffraction into street canyon (see Section 5.3) always maps one wall edge to one source point for LOS since only ray paths that hit the receiver plane are computed. The calculation of propagation paths over rooftops is different because diffraction source points lie on the whole edge of the roof, see. Figure 9(b).

Therefore, this sections describes how propagation paths due to diffraction over rooftops can be calculated efficiently on the graphics processing unit. The algorithm basically consists of two steps: (1) a discretized version of the diffraction cones is constructed for every rooftop. (2) Ray paths are found by going backwards from each receiver location towards the transmitter. An integral part of our method is the computation of diffraction beams for every rooftops simultaneously. Therefore, no identification of diffraction sources is required, we define the set of roof diffraction sources directly as the set of all building walls $\mathcal{W}$.

Let s be a radiation source. As illustrated in Figure 9(a), a *roof diffraction beam* $\mathcal{RDB}(w)$ of a wall w directly corresponds to the shadow polygon

$$\mathcal{RDB}(w) = \mathrm{NLOS}(s \rightarrow w). \tag{11}$$

Due to the construction of the beam all geometric ray paths from $\mathcal{RDB}(w)$ to the radiation source s have a deflection point somewhere on the wall w at rooftop level, e.g. $s \rightsquigarrow w \rightsquigarrow r$.

For a point $r \in \mathcal{RDB}(w)$ the exact deflection point x_r on the wall can by found be the intersection of the two lines $l_0 = (w.p_2, w.p_3)$ and $l_1 = (r, s)$, hence

$$x_r = \mathrm{intersect}(l_0, l_1). \tag{12}$$

The idea is sketched in Figure 9(b). The geometric ray paths $\mathcal{RP}(b)$ for each beam $b \in \mathcal{RDB}(w)$ are then

$$\mathcal{RP}(b) = \{s \rightsquigarrow x_r \rightsquigarrow r \mid r \in \mathcal{RDB}(w)\}. \tag{13}$$

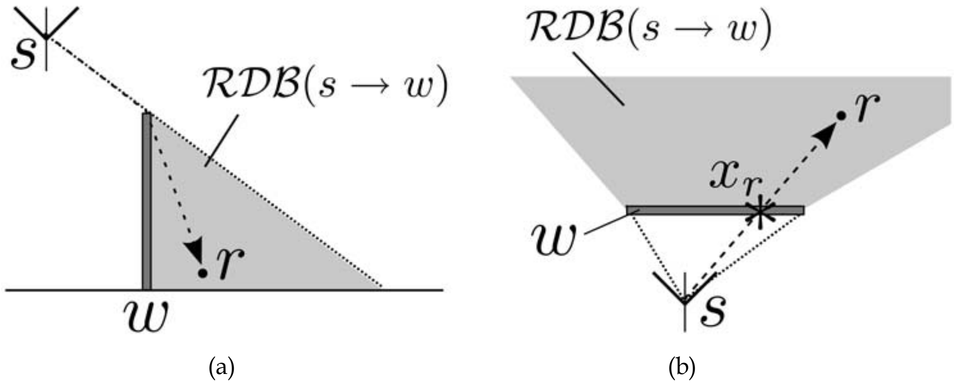


Fig. 9. Illustration of a roof diffraction beam. (a) The diffraction beam of the source s and wall w is depicted in a side profile. (b) The top view reveals the exact deflection point x_r as the intersection of the direct connection between s and r. This yields the ray path $s \rightsquigarrow x_r \rightsquigarrow r$.

The implementation of this method again involves the LOS computation on the GPU. However, some care has to be taken when ray paths are concatenated due to diffraction over multiple rooftops. If two consecutive points of a ray path are in LOS to each other, all points in between must be removed from the final path. This can be achieved by always computing the upper convex hull of all points on a ray path. Further implementation details can be found in Catrein et al. (2007).

6. Path loss calculation

This section depicts how a ray path is mapped to received signal strength. We handle the cases for LOS and NLOS separately. We have an unobstructed clear line-of-sight path if a ray path is of the form

$$\tau^{\rightsquigarrow} = s \rightsquigarrow x \tag{14}$$

for a receiver location x and a radiation source s. We write the corresponding path loss in dB as

$$PL^{\mathrm{dB}}(r) = 10\log_{10}\left(\frac{P_t}{P_r(\| x - s \|_2)}\right) \tag{15}$$

with antenna gains G, wavelength λ, transmitted power P_t and received power P_r

$$P_r(d) = \frac{P_t G \lambda^2}{(4\pi)^2 d^2} \tag{16}$$

according to the free space propagation loss by Rappaport (1995).

The path loss prediction in regions with no direct LOS is known to be more complex. Consider a ray path $\upsilon^{\rightsquigarrow}$ given as

$$\upsilon^{\rightsquigarrow} = x_{i-1} \rightsquigarrow x_i \rightsquigarrow x_{i+1} \tag{17}$$

The change of direction $\alpha_{\upsilon^{\rightsquigarrow},i}$ according to the deflection of the wave front at x_i is given by

$$\alpha_{v^{\sim},i} = \operatorname{acos}\left(\langle \| x_i - x_{i-1} \|_2, \| x_{i+1} - x_i \|_2 \rangle\right). \tag{18}$$

Let α be the change of direction of the corresponding deflection, the angle depended attenuation can then be represented by a polynomial of degree 3

$$PL(\alpha) = a_0 + a_1\alpha + a_2\alpha^2 \tag{19}$$

We retrieve the unknown model coefficients (α_0, α_1, α_2) by a calibration to real-world measurements, thus modeling the stochastic influence of traffic and vegetation. The calibration can be realized by a formulation as a constraint least-square problem. The optimal parameter vector can then be calculated by common solver algorithms like Gauss-Newton or Levenberg-Marquardt (see Levenberg (1944)).

Different types of wave guiding effects are taken into account by introducing distinct attenuation functions, the attenuation functions due to diffraction into street canyons and over rooftops are denoted by $PL^{\mathrm{dB}}_{street}(\alpha)$ and $PL^{\mathrm{dB}}_{roof}(\alpha)$. For the attenuation due to wall transmissions $PL^{\mathrm{dB}}_{wall}(n)$ we use a model similar to the Multi-Wall model by Lott & Forkel (2001) which depends on the number of penetrated walls n.

In logarithmic notation, the overall attenuation of a receiver location r is then the sum of the arriving ray paths and can be written as

$$\begin{aligned} PL^{\mathrm{dB}}_{\mathrm{NLOS}}(r) = & PL^{\mathrm{dB}}(d) + \sum_{i=1}^{N_{street}} PL^{\mathrm{dB}}_{street}(\alpha_{street,i}) \\ & + \sum_{j=1}^{N_{roof}} PL^{\mathrm{dB}}_{roof}\left(\alpha_{roof,j}\right) + PL^{\mathrm{dB}}_{wall}(N_{wall}), \end{aligned} \tag{20}$$

where N_{street}, N_{roof} and N_{wall} denote the number of deflection points and $\alpha_{street,i}$, $\alpha_{roof\ ,j}$ the corresponding changes of direction.

7. Results

In this section we present prediction accuracy and calculation time. For the purpose of benchmarking we use building and urban micro cell measurements of downtown Munich, Germany. This data has been created during the COST 231 action as described by Damosso (1999) and is now publicly available at Mannesmann Mobilfunk GmbH, Germany (1999). The scenario comprises 7 km^2 of approximately 18000 building walls. Three different measurement routes are available, referred to as METRO200, METRO201 and METRO202. All predictions were performed on the whole area of 7 km^2 with a resolution of 5 meter. This requires the evaluation of more than $3 \cdot 10^5$ receiver points. The transmitter location is depicted in Figure 10 which shows a typically field strength prediction. Colors are chosen such that diffraction at building edges is clearly visible. In the following, all path loss calculations are based on a calibration on the measurement points along route METRO202.

7.1 Computational performance

The key component of the presented algorithm is the computation of LOS regions and is therefore analyzed in more detail. We have evaluated the performance of our GPU implementation of the LOS computation on different generations of graphics cards: NVIDIA

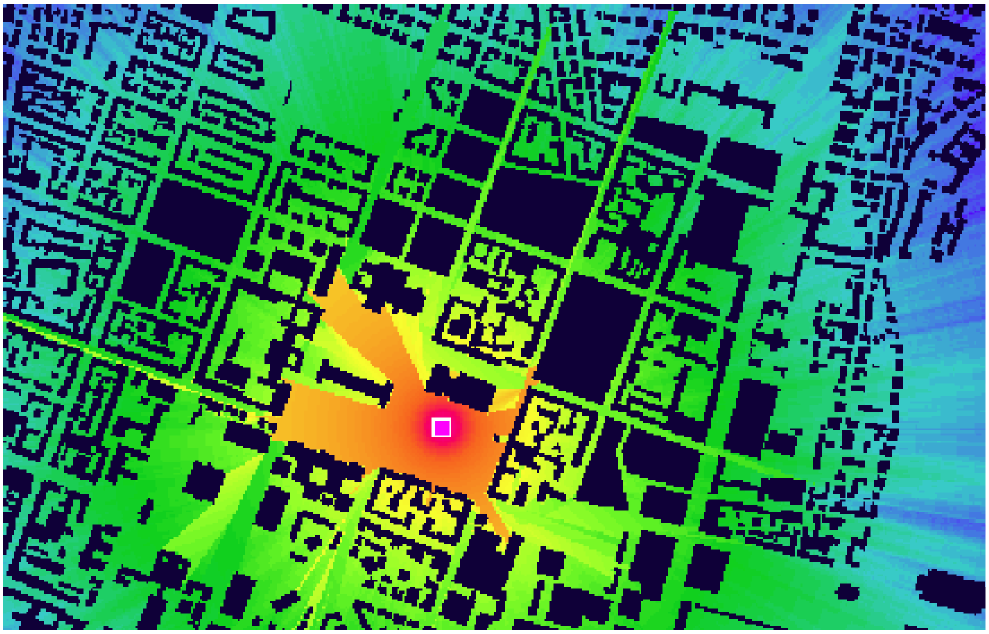


Fig. 10. Field strength prediction in downtown Munich (Germany) with colors according to intensity.

GeForce 6600 GT (May 2004), 7800 GT (August 2005) and a 8800 GTX (May 2007). Figure 11 sketches the computations per second over the size of the underlying discrete grid. At a resolution of 5 meters these cards achieve 100, 200 and over 500 LOS computations per second, respectively. Thus, we observed a speedup by a factor of 2 for every new generation of graphics cards. Hence, we conclude that the implementation of this algorithm in graphics hardware scales well on new generations of graphics hardware. In terms of throughput, the 500 LOS computations per second can also be expressed as roughly $1.6 \cdot 10^8$ receiver points per second. As point of reference, the total number of receiver points is roughly $3 \cdot 10^5$ at a resolution of 5 meter.

The implementation of the transmission depth exhibits similar performance behavior as the LOS computation since there is almost no noticeable increase in rendering load by this approach.

Table 2 gives an overview of different propagation predictions with respect to accuracy and runtime for route METRO201. We use the CPU implementation of the ray launching algorithm CORLA by Mathar et al. (2007) as reference for our GPU implementation. In our test scenario, CORLA exhibits run times of about 8 seconds. When switching from the CPU implementation of CORLA to our GPU method we observed a speedup of roughly 2.5X for the roof diffraction and a speedup of 160X for the diffraction into street canyon with additional transmission depth. The computation time of propagation paths over rooftops is about 3 seconds. The most time (2 seconds) is currently consumed by the computation of upper convex hull, since this has to be executed on the CPU, due to a missing stack implementation on the GPU.

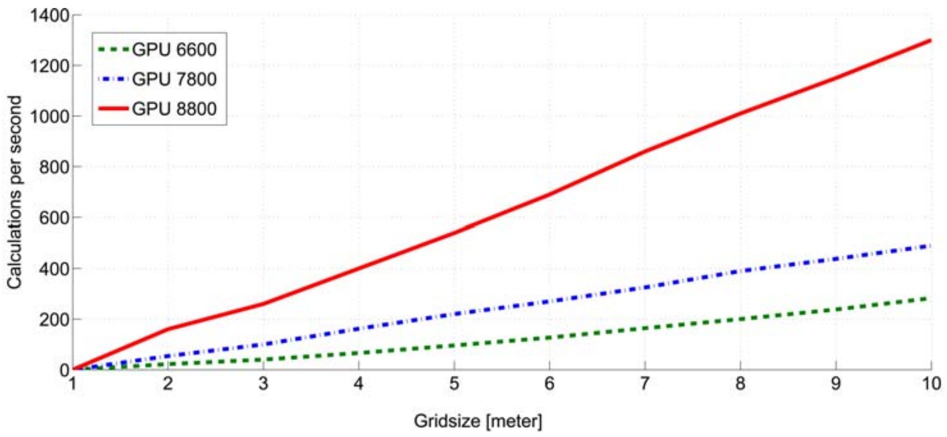


Fig. 11. Line-of-sight calculation time on different generations of graphics cards. Test scenario is COST 231 Munich.

Prediction model	mean error	std. dev.	runtime
CORLA Mathar et al. (2007) (CPU)	0.1 dB	4.2 dB	8 s
Propagation over roofs (GPU)	−0.2 dB	4.7 dB	3.05 s
Propagation into street canyons with transmission depth (GPU)	0.1 dB	4.5 dB	0.05 s

Table 2. Accuracy and runtimes of propagation models in COST 231 Munich along route METRO201.

Furthermore, we neglected the angle deviation in the propagation paths over rooftops and counted only the number of diffractions (which can be performed by the algorithm for transmission depth). In combination with the calculation of diffraction paths into street canyons, the runtime was reduced to approximately 0.05 seconds at roughly the same prediction accuracy.

7.2 Prediction accuracy

We quantify the accuracy of our propagation predictions by the *mean squared error* (mse) and the *standard deviation* (std. dev.) between prediction and measurement data. Let the number of measurements points be N, and r_i the ith measurement point. $M_{\mathrm{dB}}(r_i)$ denotes the measured, and $PL_{\mathrm{dB}}(r_i)$ the predicted path loss at r_i.
We define the mean squared error as

$$mse = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[M^{\mathrm{dB}}(r_i) - PL^{\mathrm{dB}}(r_i)]^2} \tag{21}$$

and the standard deviation as

$$std.dev. = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}[M^{\mathrm{dB}}(r_i) - PL^{\mathrm{dB}}(r_i) - \overline{x}]^2} \tag{22}$$

with mean error

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} \left[M^{\mathrm{dB}}(r_i) - PL^{\mathrm{dB}}(r_i) \right]. \tag{23}$$

A comparison between our prediction and the three available measurement routes according to these performance criteria is given in Table 3. Propagation accuracy according to the mean squared error lies between 4.5 and 6.2 dB. Whether the GPU or CPU was involved in the prediction computation had no influence on the propagation accuray as depicted in Table 2. The exact shape of the predicted and measured path loss is illustrated in Figure 12 for route METRO201. It can be seen that although the path loss is sometimes over- or underestimated, most of the important features in the measurement data are captured quite well by the propagation prediction.

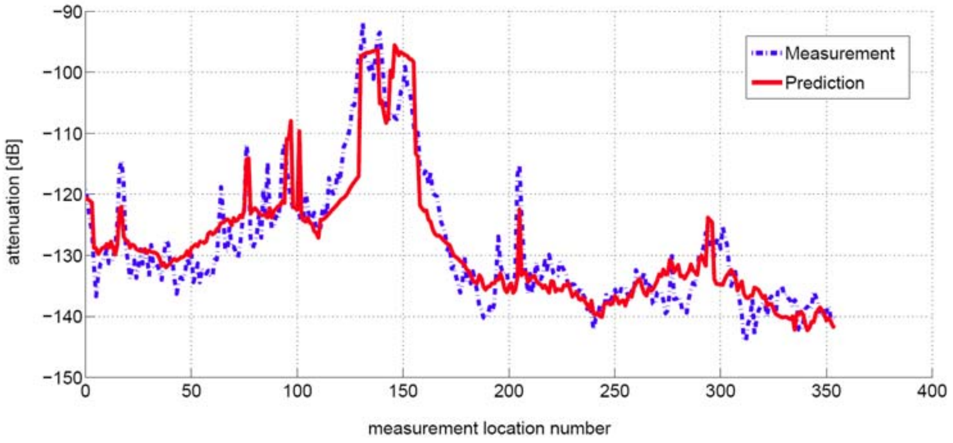


Fig. 12. Comparison between measured and predicted pathloss along route METRO201 in COST 231 Munich.

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 3. Prediction accuracy in COST 231 Munich.

8. Conclusion

In this chapter, we exploited graphics hardware for accelerating the computation of radio wave propagation predictions. Our method traces discrete line-of-sight beams and reconstructs ray paths based on radiation source, beam origin and all discrete points within a sampled beam.

All algorithms are designed to benefit from the computational power and parallel architecture of modern graphics cards. Core component of the GPU algorithms is a high

throughput LOS computation which leads to computation times of 3 to 0.05 seconds in a scenario of 7 km^2 with roughly 18000 walls. This results in a speedup of 2.5X to 160X times compared to the CPU algorithm CORLA. This is achieved by designing separate algorithms for distinct propagation effects like diffraction over rooftops and diffraction into street canyons. Hence, a proper combination of propagation effects computed on the GPU can deliver propagation predictions at interactive rates. The accuracy of our propagation predictions is quantified by a standard deviation between predictions and measurement data of 4 to 7 dB which is considered as a very good result. Thus, the use of graphics hardware for field strength predictions does not diminish propagation accuracy.
We summarize the main contribution as the development of a run-time efficient algorithm that accurately predicts mean received signal strengths in dense urban propagation environments.

A. Abbreviations and mathematical operators

LOS line-of-sight
NLOS non-line-if-sight
$\| \vec{x} \|_2$ length of vector $\vec{x}$
$\langle \vec{x}, \vec{y} \rangle$ scalar product of two vectors $\vec{x}$ and $\vec{y}$
$\wedge$ logical operator *and*
$\vee$ logical operator *or*
$\cap$ set operator *intersection*
$\cup$ set operator *union*
$\varnothing$ empty set
$x_{i-1} \rightsquigarrow x_i \rightsquigarrow x_{i+1}$ ray path that starts at x_{i-1}, changes direction at x_i and arrives at x_{i+1}

9. References

Akenine-Möller, T. & Haines, E. (2002). *Real-Time Rendering*, Natick, MA: A K Peters, Ltd.
Andersen, J., Rappaport, T. & Yoshida, S. (1995). Propagation measurements and models for wireless communication channels, *IEEE Commun. Mag.* 33: 42–49.
Carr, N. A., Hoberock, J., Crane, K. & Hart, J. C. (2006). Fast gpu ray tracing of dynamic meshes using geometry images, *GI '06: Proceedings of Graphics Interface 2006*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, pp. 203–209.
Catrein, D., Reyer, M. & Rick, T. (2007). Accelerating radio wave propagation predictions by implementation on graphics hardware, *Proc. IEEE Vehicular Technology Conference*, Dublin, Ireland, pp. 510–514.
Correia, L. M. (ed.) (2006). *COST Action 273: Mobile Broadband Multimedia Networks, Final Report*, Academic Press.
Dachsbacher, C., Stamminger, M., Drettakis, G. & Durand, F. (2007). Implicit visibility and antiradiance for interactive global illumination, *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, p. 61.

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

Erceg, V., Fortune, S. J., Ling, J., Rustako, J. & Valenzuela, R. A. (1997). Comparison of a computer-based propagation prediction tool with experimental data collected in urban microcellular environments, *IEEE J. Sel. Areas Commun.* 15: 677–684.

Erceg, V., Greenstein, L. J., Tjandra, S. Y., Parkoff, S. R., Gupta, A., Kulic, B., Julius, A. A. & Bianchi, R. (1999). An empirically based path loss model for wireless channels in suburban environments, *IEEE J. Sel. Areas Commun.* 17: 1205–1211.

Fernando, R. (2004). *GPU Gems*, Addison-Wesley, chapter Effective Shadow Volume Rendering, pp. 137–166.

Fernando, R. & Kilgard, M. (2003). *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics,* Reading, MA: Addison-Wesley.

Funkhouser, T., Min, P. & Carlbom, I. (1999). Real-time acoustic modeling for distributed virtual environments, *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 365–374.

Hata, M. (1980). Empirical formula for propagation loss in land mobile radio services, *IEEE Trans. Veh. Technol.* 29: 317–325.

Heckbert, P. S. & Hanrahan, P. (1984). Beam tracing polygonal objects, *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, pp. 119–127.

Horn, D. R., Sugerman, J., Houston, M. & Hanrahan, P. (2007). Interactive k-d tree gpu raytracing, *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, pp. 167–174.

Ikegami, F., Takeuchi, T. & Yoshida, S. (1991). Theoretical prediction of mean field strength for urban mobile radio, *IEEE Trans. Antennas Propag.* 39: 299–302.

Ikegami, F., Yoshida, S., Takeuchi, T. & Umehira, M. (1984). Propagation factors controlling mean field strength on urban streets, *IEEE Trans. Antennas Propag.* 32: 822–829. Intel Corporation (2008). URL: *http://www.intel.com*

Kajiya, J. T. (1986). The rendering equation, SIGGRAPH '86: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques,* ACM Press, New York, NY, USA, pp. 143–150.

Kim, S.-C., Guarino, B. J., III, T. M.W., Erceg, V., Fortune, S. J., Valenzuela, R. A., Thomas, L.W., Ling, J. & Moore, J. D. (1999). Radio propagation measurements and prediction using three-dimensional ray tracing in urban environments at 908 mhz and 1.9 ghz, *IEEE Trans. Veh. Technol.* 48: 931–946.

Levenberg, K. (1944). A method for the solution of certain problems in least squares, *Appl. Math.* 2: 164–168.

Lott, M. & Forkel, I. (2001). A multi-wall-and-floor model for indoor radio propagation, *Proc. IEEE Vehicular Technology Conference*, Vol. 1, Rhodes, Greece, pp. 464–468.

Mannesmann Mobilfunk GmbH, Germany (1999). Cost 231 - urban micro cell measurements and building data.
URL: *http://www.ihe.uni-karlsruhe.de/forschung/cost231/cost231.en.html*

Mathar, R., Reyer, M. & Schmeink, M. (2007). A cube oriented ray launching algorithm for 3d urban field strength prediction, *IEEE International Conference on Communications*, pp. 5034 – 5039.

Overback, R., Ramamoorthi, R. & Mark,W. R. (2007). A real-time beam tracer with application to exact soft shadows, *EGSR*.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E. & Phillips, J. C. (2008). GPU computing, *Proceedings of the IEEE* 96(5): 879–899.

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krger, J., Lefohn, A. E. & Purcell, T. J. (2005). A survey of general-purpose computation on graphics hardware, *Eurographics 2005, State of the Art Reports*, pp. 21–51.

Rajkumar, A., Naylor, B. F., Feisullin, F. & Rogers, L. (1996). Predicting rf coverage in large environments using ray-beam tracing and partitioning tree represented geometry, *Wirel. Netw.* 2(2): 143–154.

Rappaport, T. S. (1995). *Wireless Communications: Principles and Practice,* Prentice-Hall, Inc.

Rick, T. & Mathar, R. (2007). Fast edge-diffraction-based radio wave propagation model for graphics hardware, *Proc. IEEE 2nd International ITG Conference on Antennas*, Munich, Germany, pp. 15–19.

Schaubach, K. R., IV, N. J. D. & Rappaport, T. S. (1992). A ray tracing method for predicting path loss and delay spread in microcellular environments, *Proc. IEEE Vehicular Technology Conference*, Vol. 2, pp. 932–935.

Schmitz, A. & Kobbelt, L. (2006). Wave propagation using the photon path map, *PE-WASUN '06*, ACM, New York, NY, USA, pp. 158–161.

Schmitz, A., Rick, T., Karolski, T., Kuhlen, T. & Kobbelt, L. (2009). Simulation of Radio Wave Propagation by Beam Tracing, *Eurographics Symposium on Parallel Graphics and Visualization*, pp. 17–24.

Shreiner, D., Neider, J., Woo, M. & Davis, T. (2003). *OpenGL Programming Guide*, fourth edn, Reading MA: Addison-Wesley.

Stam, J. (1999). Diffraction shaders, *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 101–110.

Tsingos, N., Funkhouser, T., Ngan, A. & Carlbom, I. (2001). Modeling acoustics in virtual environments using the uniform theory of diffraction, *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, pp. 545–552.

Tsingos, N., Gallo, E. & Drettakis, G. (2004). Perceptual audio rendering of complex virtual environments, *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, pp. 249–258.

Wahl, R., Wlfle, G., Wertz, P., Wildbolz, P. & Landstorfer, F. (2005). Dominant path prediction model for urban scenarios, *14th IST Mobile and Wireless Communications Summit*.

Whitted, T. (1980). An improved illumination model for shaded display, *Commun. ACM* 23(6): 343–349.