Efficient Rasterization for Outdoor Radio Wave Propagation

Arne Schmitz*, Tobias Rick+, Thomas Karolski*, Torsten Kuhlen+ and Leif Kobbelt*
*Computer Graphics Group, +Virtual Reality Group, RWTH Aachen University

(Invited Paper)

Abstract—Conventional beam tracing can be used for solving global illumination problems. It is an efficient algorithm, and performs very well when implemented on the GPU. This allows us to apply the algorithm in a novel way to the problem of radio wave propagation. The simulation of radio waves is conceptually analogous to the problem of light transport. We use a custom, parallel rasterization pipeline for creation and evaluation of the beams. We implement a subset of a standard 3D rasterization pipeline entirely on the GPU, supporting 2D and 3D framebuffers for output. Our algorithm can provide a detailed description of complex radio channel characteristics like propagation losses and the spread of arriving signals over time (delay spread). Those are essential for the planning of communication systems required by mobile network operators. For validation, we compare our simulation results with measurements from a real world network. Furthermore, we account for characteristics of different propagation environments and estimate the influence of unknown components like traffic or vegetation by adapting model parameters to measurements.

Index Terms—Ray tracing, Rendering, Electromagnetic propagation

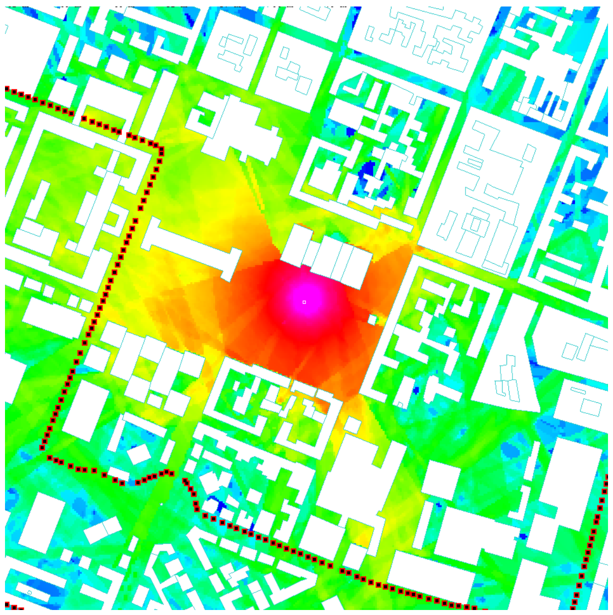
Fig. 1. Visual result for the Munich scenario, marking the measurement route.

1 Introduction

Global illumination deals with the propagation of light. It provides the basis for various image synthesis algorithms and leads to convincing, realistic results due to its physical correctness. Important propagation phenomena are reflection and refraction. Effective and efficient algorithms for solving the global illumination problem are long known [1]. Typical algorithms are often based on ray tracing techniques.

Visible light occupies only a small fraction of the electromagnetic spectrum. The propagation of waves in other frequency ranges like those of sound or radio waves behave similar to the propagation of light. Waves propagate as fronts and spread in different directions when interacting with obstacles. However, since wavelengths of sound and radio are in size similar to the environment (up to several meters), the effect of waves bending around corners (diffraction) becomes important. Telecommunications systems for instance use carrier frequencies with wavelengths ranging from several meters to kilometers. Noticeable delays or echoes are produced when multiple wavefronts with different travel times due to multiple propagation paths arrive at the same location. This effect is recorded in so called delay spread histograms.

In this paper we present three main contributions. (1) A subset of a rasterization pipeline has been implemented purely in software on the CUDA platform. We did this, since the OpenGL pipeline does not offer the flexibility needed to compute the 3D delay spread histograms. The presented pipeline can target arbitrary 2D and 3D framebuffer objects. (2) We developed a novel method for the efficient computation of complex radio channel characteristics by combining concepts of both electromagnetic principles and computer graphics. We apply and extend common principles of computer graphics, such as beam tracing, rasterization and general purpose GPU programming to simulate effects like diffraction that have a significant influence on the propagation behavior of radio waves. (3) We present a scheme for adapting model parameters from real-world

This work has been supported by the UMIC Research Centre, RWTH Aachen University and the Jülich-Aachen Research Alliance for Simulation Science JARA-SIM.

measurements to account for characteristics of different propagation environments and to estimate the influence of unknown components like traffic or vegetation.

We target the propagation of radio waves of common mobile communication systems, which are in the range of several hundred MHz up to a few GHz. The knowledge of detailed radio channel characteristics is an essential requirement in the design of future communication systems. Multiple propagation paths give rise to signal fluctuations and additional propagation losses. A receiver has to deal with the arrival of signals coming from multiple directions which are spread over time. The so-called delay spread is therefore of great interest in order to provide lower and upper bounds for the lengths of incoming signals to avoid inter-symbol interference.

Both global illumination and radio wave propagation algorithms often rely on ray tracing techniques. However, high quality solutions often come at the cost of excessively high computation times. The prediction of the propagation behavior of radio waves in a common urban scenario can take from minutes up to hours. Excessive computation times still prevent the large scale deployment of exact propagation algorithms in the wireless communications community since a huge number of computations are required for the simulation of radio networks.

Thus, reducing the computation time of radio wave propagation simulations is an important topic, which we will discuss in this work. We show how to compute the delay spread histogram in an efficient and accurate manner, which was not possible before. We obtain a significant speedup by the following concept. Rather than tracing individual rays, the use of beams has the advantage that less undersampling occurs, since the beams represent a continuous bundle of propagation rays, which was also noted by Lehnert [2]. Furthermore, we implemented the algorithm on the NVIDIA CUDA platform, since our algorithm is highly parallel in nature and as thus can utilize massively parallel compute platforms. There are a number of similar technologies emerging, apart from CUDA. For example the Larrabee platform is conceptually similar as shown in [3]. On that particular system the OpenGL pipeline is done entirely in software. Our approach uses a similar technique, implementing a subset of a standard rasterization pipeline.

The basic idea of our algorithm is to generate beams that emanate from a radiation source, and split those beams by a rasterization step, recursively creating new secondary beams. Beams are evaluated by a second rasterization step, which produces a 2D field strength map and a 3D delay spread map. All raster operations are implemented solely in CUDA, utilizing the parallel nature of the platform.

Common input data for radio wave propagation algorithms is a building database where a building is usually described by its polygonal outline and one height value for the roof. We refer to this description as 2.5 dimensional. The building heights actually influence the wave propagation, since visibility is determined according to the height of the transmitter. The restriction to two dimensions is a common abstraction in wave propagation simulations, and helps to reduce the computational complexity by a large amount. However, this model also has some obvious drawbacks. For this assumption to be correct, the geometry of the urban terrain has to be more or less planar. Predictions in mountainous regions will most likely be wrong. Also, this approach makes it impossible to simulate multi-story buildings in an indoor scenario.

Hence for a more general framework we plan to extend the method to 3D beam tracing. The extension to 3D leads to new challenges. Especially traversing the scene geometry, producing a continuous reflection and diffraction field, and the more complex rasterization are interesting problems. Some of these problems have already been solved in computer graphics, e.g. in the work by Overbeck et al. [4].

The presented work is an extension of our paper presented at the EGPGV [5]. Compared to our previous work, we have improved the performance of the rasterization and beam splitting process, added a method to optimize the propagation parameters from measurements, and we added another real-world scenario, namely a setup of three base stations in the city of Ilmenau with accurate measurements.

The remainder of this paper is organized as follows. After reviewing previous work on global illumination and radio wave propagation in Section 2, we give an overview of our algorithm in Section 3. We then present details of our parallel rasterization, how reflected and diffracted beams are generated and traced, and how the attenuation of the electromagnetic radiation is finally derived in Section 4. After that, we introduce a scheme for adapting model parameters by formulating a constraint least squares problem in order to minimize the mean squared error between predicted and measured data in Section 5. We finally give a detailed analysis of performance and accuracy in Section 6 and conclude the paper in Section 7.

2 Related Work

Classical ray tracing was introduced by Whitted [6]. Since then it has been successfully applied and extended in numerous publications in order to compute global illumination effects based on geometrical optics. There are various publications that focus on mapping global illumination algorithms onto the GPU, which include, but are not limited to, Horn [7], Carr [8] and Dachsbacher [9]. Global illumination techniques have been used for different problems before, e.g., for sound rendering. Notable here are the works of Tsingos [10], [11] and Funkhouser [12].

Global illumination and radio wave propagation are essentially the same problem statement, although they differ slightly in the kind of optical effects that are simulated. Diffraction and interference for example are usually left out of global illumination, due to the subtlety of the effect. But some works like Stam [13] and Tsingos et al. [10] do incorporate these effects. However, the basic rendering equation, as formulated by Kajiya [1], still holds for radio wave propagation and can be used almost as is.

The theoretical foundation of radio wave propagation can be found in the book of Rappaport [14] whereas the COST 273 report [15] gives a more recent overview on radio propagation models and algorithms. In literature, it is very common to distinguish between stochastic (empirical) channel models and deterministic propagation algorithms. Well-known examples of empirical models are the work of Hata [16] and Ikegami [17]. They propose to model the radio propagation phenomena by approximating the actual propagation loss (path loss) by parametrized functions. Hata determined the values of the parameters by conducting extensive measurement campaigns. Ikegami extended Hata's work by analyzing the dependence of approximate equations with respect to height gain, street width, propagation distance and radio frequency. Such 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 [18].

Therefore, most deterministic algorithms for predicting radio signal strength rely on the computation of actual propagation paths due to wave guiding effects like reflection, diffraction and scattering. Ikegami [19] showed that ray tracing is also an excellent technique for estimating radio propagation losses. Based on ray tracing algorithms, Schaubach [20], Schmitz [21] and Kim [22] state that their predicted path loss values were generally within 4 to 8 dB of the measured path loss. Such predictions are considered to be of very high accuracy.

The idea of ray tracing can be extended to the concept of beams, which are a continuum of rays. Beam tracing was introduced by Heckbert and Hanrahan [23]. It reduces intersection tests, as well as overcomes sampling problems, since ray samples tend to become too sparse or too dense. Many more works have been published in this area, which also concentrated on real-time rendering [4], or non-graphical applications such as audio rendering. Two examples for this are the works by Funkhouser et al. [12] and Chandak et al. [24].

However, our approach takes the beam tracing idea to a different level of applications, not simulating light, but the radiation of different radio frequency bands. In combination with our novel data structures and an efficient implementation using general purpose GPU programming, this allows us to calculate both field attenuation and delay spread at the same time. This is done with high accuracy and in a speed not possible before. Similar to our approach is the work by Rajkumar et al. [25], who also used a form of beam tracing for wave propagation, but determines visibility differently. Similarly, Fortune presented a beam tracing approach for indoor wave propagation [26].

Furthermore, Rick et. al. presented a GPU-based approach to radio wave propagation in Catrein [27] and Rick [28]. They trace propagation paths in a discrete fashion by repeated rasterization of line-of-sight regions. By restricting computations to the strongest path only, propagation predictions are delivered at interactive rates. However, since only the mean received signal strength is computed, multi-path effects, which are an essential requirement for delay spread estimations, are completely neglected. This is not the case with our algorithm. Besides basic propagation losses, advanced channel characteristics like the delay spread are computed at a considerably reduced run time.

3 Overview

Our approach rapidly and accurately computes two important aspects of radio wave propagation at arbitrary points in the scene: the average field strength and a delay spread histogram.

The algorithm consists of two parts: First, it builds a beam hierarchy that describes the propagation of the electromagnetic radiation, and second it evaluates the radio field properties based on this beam hierarchy.

The tracing algorithm relies on a small rendering pipeline similar to OpenGL, but implemented in CUDA, to determine the split positions inside of each beam. For efficiency, unnecessary geometry is clipped away by the use of a quadtree that is intersected with the beam. The pseudo code of the tracing algorithm is as follows:

```
1. Build scene geometry quadtree
2. Trace initial beams from source
  1 Clip scene against beam using quadtree
  2 Split beam according to visible
    geometry
  3 Generate reflected, refracted and
    diffracted beams
  4 Update signal time and attenuation
    for beam
  5 Trace recursively
```

The evaluation of the generated beams also uses a simplified rasterization pipeline, which accumulates the beam attenuation and the delay into 2D and 3D framebuffers respectively. The pseudo code of the evaluation algorithm using the information computed in step 2.4 in the previous listing is thus:

```
1. Iterate over all beams
  1 Rasterize beam attenuation into
    2D array
  2 Rasterize beam delay into
    3D histogram
```

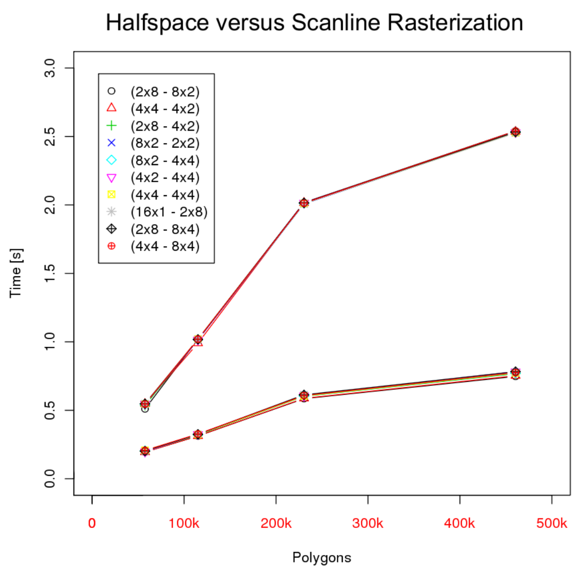

Fig. 2. Performance of the ten best grid configurations for each rasterization algorithm. The top graphs represent the half-space rasterization algorithm, while the bottom graphs represent the scanline rasterization.

4 The Algorithm

4.1 Parallelized Rasterization

The following sections describe the beam tracing algorithm, which heavily relies on our rasterization engine. It is a subset of a standard rendering pipeline and it is implemented in CUDA. The pipeline consists of transformation, clipping and rasterization steps. All steps are implemented in software only, and are fully parallelized. The advantage of implementing our own rendering pipeline, instead of using e.g. OpenGL, is the increased flexibility. Our engine supports framebuffers of arbitrary size, both 2D and 3D. Especially the latter property allows us to efficiently generate delay spread histograms for every point in the scene. With the advent of many-core architectures on GPUs, which are replacing fixed-function graphics pipelines, implementing rendering pipelines in software is again becoming an interesting topic. Hardware such as the Intel Larrabee platform implement rasterization not in hardware anymore, but perform it in software. This gives increased control over the details of the implementation of the rasterization pipeline and is why we decided to investigate this in more detail.

We have implemented two rasterization algorithms so far to test their performance on massively parallel platforms. One employs a simple scanline-conversion. The other algorithm uses halfspace-rasterization, as described by Pineda [29]. Both algorithms have their advantages and disadvantages concerning code complexity and performance. In our experiments the half-space rasterization algorithm turned out to be about twice as fast, which is why we used it in the final evaluation.

4.1.1 Scanline Rasterization

Many of the earlier methods of converting a polygon to pixels are methods based on scanline conversion. Our implementation is based on the scanline rasterization algorithm by Heckbert [30]. The algorithm divides the rasterization task into horizontal scanlines using increments from line to line to adjust the start and end point of the raster line. The increments are updated on the fly at each vertex of the boundary. There are two increments dl and dr for the left and the right side of the polygon boundary.

This on the spot updating of the parameters is not well suited for massively parallel platforms like GPUs. It results in lots of additional random memory accesses, slowing down the execution time considerably.

Instead of this we decided to precompute the events where a new vertex is encountered and put all necessary rasterization parameters per polygon into an array. This can then be laid out nicely aligned so that memory access becomes sequential and coherent over all threads. Moreover, we store the four increment parameters in 2D textures, since it is the fastest way to access memory on current GPUs. Texture memory is usually the only memory on a GPU equipped with a small cache to buffer access to it. So we just store for each vertex encountered which of the increment variables has to be updated, as well as the new scanline start and end position. The parallelization of the algorithm can happen both inside of a single scanline as well as over several scanlines at once, up to the next vertex event. For coherent memory accesses that are well aligned between all threads, we use a simple tiling mechanism that covers the polygon.

4.1.2 Half-Space Rasterization

A somewhat more advanced algorithm uses half-space classification to determine if a pixel lies inside or outside of the triangle. Such algorithms were introduced first by Pineda [29], but many more and newer variations of this exist. The advantage lies in the more efficient parallelization of these algorithms, since the half-space tests can be performed for a large number of pixels in parallel. Another advantage is that when whole pixel blocks are processed, trivial accept and reject cases can be used, to perform the rasterization even faster. Only blocks that are partially covered by the triangle need closer inspection.

However this class of algorithms comes with a somewhat higher computational cost. More registers are needed for storing variables and temporary results. This means not as many threads can be launched as with the scanline algorithm. But as we will show in the next section, it still pays off, since this class of algorithms seems to fit the GPU model better than scanline based algorithms.

4.1.3 Performance evaluation

The advantage of the scanline rasterization is that the algorithm is rather simple and small. Thus it uses relatively few registers, which are an important and valuable resource on current GPUs. For example NVIDIA GeForce GPUs usually have either 8,192 or 16,384 scalar registers available per multiprocessor. This limits either the amount of usable registers, or the number of threads that can be launched concurrently on a multiprocessor. The more threads can be launched, the higher the utilization of the GPU. However, the scanline based rasterization algorithm needs a pre-processing step to compute the vertex events and their respective parameters. This leads to reduced performance compared with the half-space rasterization algorithm.

Block based algorithms like the half-space test tend to map better to the GPU architecture. Current GPUs and GPU based compute APIs use a grid computing approach (e.g. NVIDIA CUDA or OpenCL [31], [32]). A grid consists of blocks, which in turn consist of discrete threads. Blocks can be one-, two- or three-dimensional, which can be used to easily access thread-specific data in memory.

We performed a thorough benchmarking of both rasterization algorithms, by testing them with a variety of block and grid configurations. This is shown in Fig. 2, where it can be seen that the half-space algorithm clearly outperforms the scanline algorithm by more than a factor of two. We plotted the ten best grid configurations, which for both algorithms are very close together. The block sizes are often 16 or 32, which is natural for CUDA devices, since the multiprocessors contain so called half-warps of size 16 which run in lock-step. I.e. threads in those blocks have to run exactly the same instructions. So it is very useful to group threads in sizes of 16 or 32 together that perform locally similar code paths, which is the case if they rasterize neighboring pixels.

It is still a great challenge to implement a high performance rasterization engine. Our own implementation is still at least one order of magnitude slower than the one found on current graphics hardware. But we consider our pipeline to be experimental anyway, since it supports even 3D framebuffers and everything is reprogrammable.

4.2 Beam Tracing

A beam in 2D is defined as a quadrilateral. It represents a bundle of rays emanating from an edge, which might be degenerate in the case of a point radiation source, in which case the beam is equivalent to a triangle. Each beam carries information about signal travel time, for later evaluation of the path loss and delay spread. Our algorithm generates recursively a beam hierarchy, beginning at a radiation source. Beams are reflected, refracted and also diffracted at surface boundaries. A beam might be infinite at one end, if it does not intersect any geometry at all, or it might be finite, if it intersects the geometry of the scene.

The beam is constructed from four edges (compare Fig. 3). The two edges e_1 and e_2 form the beam-cone. If the beam is created due to a reflection on a building wall, then it has another edge i that coincides with the wall. We call this the image plane of the beam. This will actually be the image plane of the viewing frustum during the beam splitting step. The last edge lies virtually at infinity for beams that do not get split, or it coincides with a part of a wall for beams that have hit some part of the geometry.

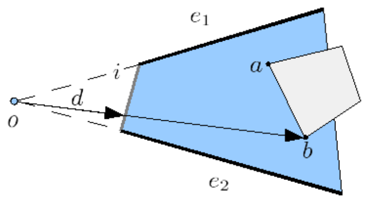

Fig. 3. A beam is defined by four edges. The two edges e_1 and e_2 form a quadrilateral whose baseline i we call the image plane of the beam. The point o is the (virtual) origin of the beam. A ray $r = o + \lambda d$ is constructed through the image plane and intersected with the face $\{a, b\}$ which was identified in the beam framebuffer.

4.2.1 *Clipping Geometry Against the Beam*

Our approach makes use of two primary data structures. A quad tree for accelerating the intersection of beams with the scene geometry, and the beam hierarchy, which describes the propagation paths the electromagnetic radiation may take.

When a beam is formed, it has to be intersected with the scene geometry in order to recursively spawn new beams that in turn will form reflected and transmitted beams. The first step of the intersection is done by enumerating all quad-tree nodes that the beam overlaps with. This helps to cull away all unnecessary geometry from the computation. Only geometry from overlapping quad-tree nodes is used in the following beam-splitting process.

4.2.2 *Splitting the Beam*

For fast and easy splitting of the beam into new sub-beams, we simply render the geometry contained in the beam into a 2D frame buffer, containing IDs for the geometric faces and an associated depth buffer. Since beams tend to become very thin after a small number of interactions, only few faces have to be rasterized.

When the faces that are hit have been identified, one has to compute exact intersection points of the beam with the wall. Given a beam as in Fig. 3, we have a ray $r = o + \lambda d$ from the beam origin $o = (x_o, y_o)$ with direction $d = (x_d, y_d)$ to the first rasterized intersection point of the beam with a face, and finally the face $f = \{a, b\}$ with points $a = (x_a, y_a)$ and $b = (x_b, y_b)$. We can now compute the intersection point $p = (x, y)^T$, by solving the two following equations:

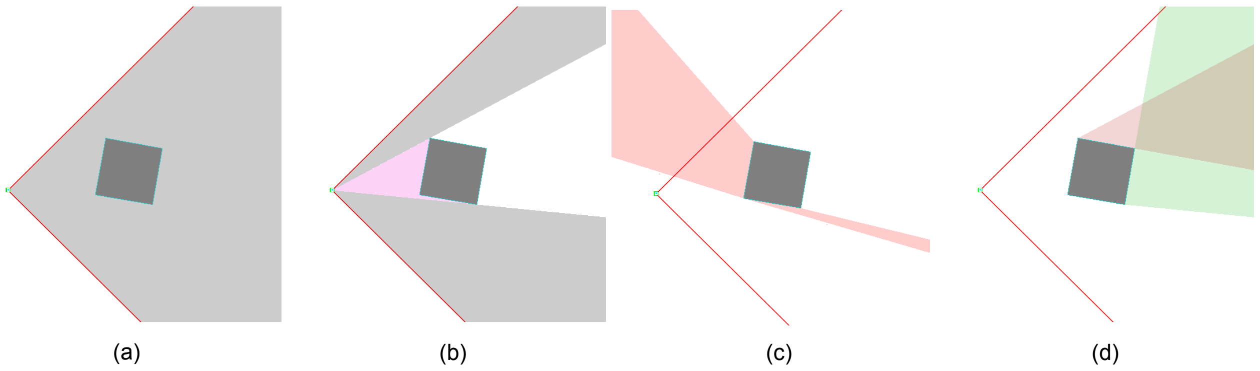

Fig. 4. (a) A beam is created and intersected with the geometry. (b) The beam is split into child-beams, according to the intersections. (c) Reflection edges are identified, and the old beam origin is reflected at those edges, constructing reflected beams. (d) In the same manner diffraction beams are generated at silhouette edges.

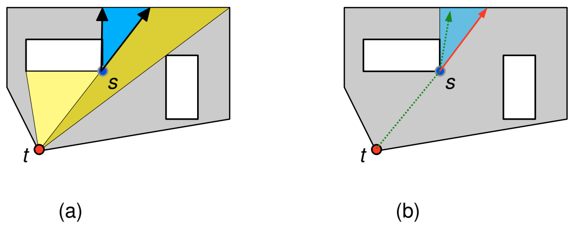

Fig. 5. (a) Diffraction beam in blue and (b) propagation path due to diffraction at building edge in green.

$$\begin{vmatrix} x & y & 1 \\ x_o & y_o & 1 \\ x_o + x_d & y_o + y_d & 1 \end{vmatrix} = 0 \tag{1}$$

$$\begin{vmatrix} x & y & 1 \\ x_a & y_a & 1 \\ x_b & y_b & 1 \end{vmatrix} = 0 \tag{2}$$

We ensure that the resulting intersection point lies on the face being intersected, by clamping the newly created beam to the endpoints of the face being tested.

4.2.3 *Generating Reflected and Diffracted Beams*

If a certain recursion depth is not yet reached, we recursively generate reflected and diffracted beams, leading to a beam tree. The reflected beam is very easily constructed by mirroring the beam origin at the wall that the old beam has hit. This is shown in Fig. 4. The new beam originates from a virtual source that is constructed by simply mirroring the source of the parent beam at the reflecting face.

Diffraction beams are constructed at silhouette edges of the geometry for every parent beam that has been split at one such edge. The diffraction beam spans the area between the shadow boundary of the parent beam and the backfacing part of the silhouette, see Fig. 5.

According to the *Geometrical Theory of Diffraction* [33] diffracted rays are produced by incident rays which hit edges, corners or vertices of boundary surfaces. Diffracted wave fronts can enter the shadow regions of the objects that are the point of diffraction. The electromagnetic field is assigned to diffracted rays similarly to reflected or transmitted rays. The initial value of the field on a diffracted ray is obtained by multiplying the field of the incident ray by a diffraction coefficient. The actual values are determined by the incident direction and the diffraction direction, wavelength and physical properties (material) of the media at the point of diffraction.

Ray tracing algorithms for radio wave propagation commonly model diffraction effects by tracing a multitude of rays into the respective diffraction cones. This fits well into the concept of beams, see Fig. 5.

Note that the beam tree contains two different kinds of beams. At every even-numbered level in the tree (starting at level 0), it contains a beam that was spawned by means of reflection or diffraction. Those beams are evaluated later if and only if they are a leaf in the tree. If, on the other hand, they have child beams, they have been split. On the odd levels of the tree, there are only beams that were produced by the splitting algorithm. Those beams have to be evaluated always, since they contain sufficient visibility information to represent a valid part of the propagation path. This gives a simple rule for the second part of the algorithm, where the beams are evaluated and the path loss is computed, see Fig. 6 for an illustration.

4.3 Beam Evaluation

4.3.1 *Computing the Attenuation*

From global illumination we know the Bidirectional Reflectance Distribution Function (BRDF) denoted by the symbol f_r. It also exists in the context of radio wave propagation. However, measured BRDFs for wavelengths on the order of centimeters are not known. But many objects and materials in this frequency spectrum are of specular nature anyhow, as stated by Rappaport [14]. So we just need to have a scene specific extinction coefficient, which can be estimated or guessed from sparse measurements.

For radio wave propagation simulations, usually the path loss is computed, which is the attenuation of the signal. The most simple model for this is the freespace model, which expresses the path loss in dB:

$$L_{\text{fp}} = 10n \log_{10} d + C \quad (3)$$

For the path loss exponent $n = 2$ and the system loss constant $C = 0$ we get the same attenuation ($\Phi \sim \frac{1}{d^2}$), that is known from global illumination for the attenuation of the flux density with the distance to the light source. In Section 4.3.2 we will further explain how to use this information to compute the final path loss at a specific point.

Our method currently does not support the computation of small-scale (spatial) fading effects due to multipath propagation. This effect is also called destructive interference and is most often modeled together with temporal small-scale fading effects, as described by Rappaport [14]. Although constructive and destructive interference of multipath propagation could probably be approximated by our method, we did not pursue this any further. It depends on the base frequency and the bandwidth of the channel used. In most systems, both spatial as well as time-varying small-scale fading will be simulated by statistical methods, on top of conventional propagation simulations such as ours. Furthermore, the geometric model underlying the simulation would need to be much more complex, including cars, scattering objects, such as trees, and possible much more detailed building models. This will be in most cases prohibitively expensive, both in terms of computational complexity as well as in resources needed to create the model.

4.3.2 Evaluating the Beam Hierarchy

After the beam tree has been generated, it has to be evaluated, to generate the final signal strength image, and to compute the delay spread histogram. This is done by traversing the beam tree and rasterizing each beam, accumulating the results in two buffers. See Fig. 6 for an illustration of the beam tree. Only beams that were created in a split step and those that are leafs of the tree will be rendered and contribute to the final field strength.

The first buffer is a 2D buffer that accumulates the signal strength, and the second buffer is a 3D buffer that contains the delay spread histogram. The algorithm rasterizes each beam, and computes the attenuation and the travel time inside of it. The attenuation is used for computing the final signal strength, and it propagates through the beam-hierarchy and through each beam with the accumulated distance from the radiation source.

Given the set B of all beams of the beam tree that lead to an individual beam b, we compute the path loss $L_p(x)$ of a point $x \in b$. We accumulate the BRDFs f_r for each reflection and also take the length of the propagation path into account. It is defined by the distance from x to the transmitter origin t_o (equivalent to the $\frac{1}{d^2}$ from global illumination). Hence, the path loss is then defined as:

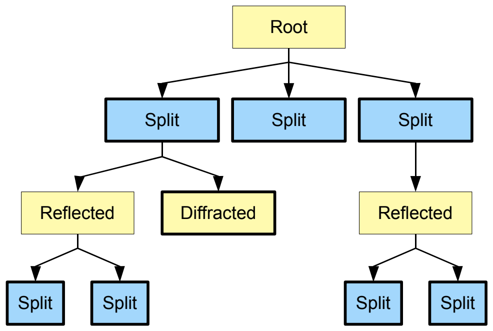

Fig. 6. The beam tree structure. Only the bold boxes will be actually rendered in the final rasterization step.

$$L_p(x) = \frac{\prod_{b_i \in B} f_r^i}{|x - t_o|^2} \quad (4)$$

This value can be written directly into a 2D array or framebuffer, using efficient polygon rasterization, implemented on the CUDA device. The accumulated path loss is computed by simply summing all beams into the 2D buffer.

4.3.3 Computing the Delay Spread

The path loss and the delay spread are computed at the same time. We use a 3D array for collecting the histogram. Every column in this array represents a discrete delay spread histogram for the given 2D position. When we evaluate (i.e. rasterize) a beam, we compute for each pixel the distance $d = |x - t_o|$ the signal has traveled so far from its virtual source t_o. This can be mapped to a traveling time, which is approximately $t = d \cdot c$, where $c \approx 299 \cdot 10^6 \frac{m}{s}$, depending on the optical density of the material. This is then mapped to one of the bins of the histogram and the path loss is added to that bin. Accumulating over all beams gives an accurate estimation of the delay spread at this specific point of the scene.

In our implementation the user can specify all relevant simulation parameters, such as the maximum recursion depth, the transmitter and receiver position, and resolution of the resulting simulation. Feedback is given on the spot, and is often interactive, depending on the scene complexity and number of reflections and diffractions. The evaluation of the delay spread is obviously instantaneous for a static transmitter and a moving receiver, and allows the user to intuitively explore the temporal spreading characteristics of the signal.

5 Model Parameter Adaption

The modeling of urban propagation environments often consists of polygonal building outlines with one height

$$
\begin{array}{cccccccccccc}
 & & \text{Distance path 1} & & & & & \text{Distance path 2} & & & & \\
M = & 1 & \log d_1^1 & 1 & 1 & 1 & 0 & \log d_1^2 & 1 & 1 & 0 & 0 & \cdots \\
 & 1 & \log d_2^1 & 1 & 1 & 0 & 0 & \log d_2^2 & 1 & 1 & 1 & 1 & \cdots \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\
 & \text{Constant} & & & \text{Prop. Path 1} & & & & & \text{Prop. Path 2} & & &
\end{array}
$$

Fig. 7. Each row of the optimization matrix corresponds to one measurement location. Each column is formed by the travel distance and number of reflections and diffractions of the arriving beams at the respective location.

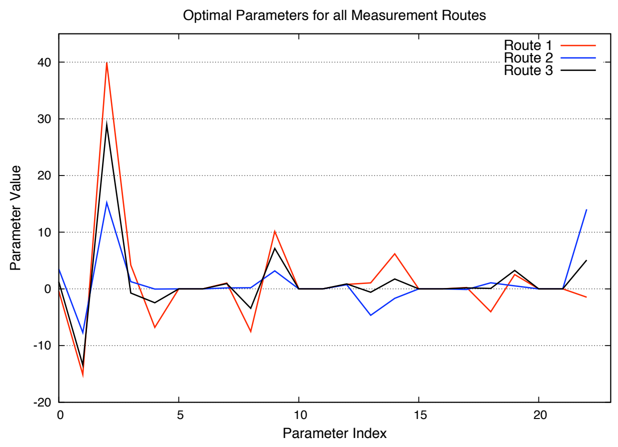

Fig. 8. Optimal parameter vectors for each measurement route.

value per building, the so-called 2.5 dimensional description. Other influencing factors like building material, roof style, texture or vegetation are typically not included in the description of the urban models since the acquisition of this information is either very expensive or sometimes simply not available. However, a city with modern skyscrapers that consist predominantly of glass fronts and flat roof tops will certainly exhibit a different attenuation behavior than a small town with pitched roofs and fronts made of concrete.

In literature, it is therefore quite common to adapt propagation models to different types of environments. A qualitative description of a propagation environment would for instance consist of a classification into rural or urban, or by building density and street widths. However, such coarse grain classification usually reflects typical propagation characteristics very poorly. Therefore, we use an implicit description by adapting model parameters to different environments by calibration from real-world measurements. Thus, we model unknown components of the propagation environment like traffic or vegetation by introducing variable coefficients (model parameters) into our path loss calculation.

Since the logarithm does not change the basic behavior of a function and therefore preserves the minimum, we choose to optimize the logarithm of the path loss formula (4). Hence, the product in the numerator is

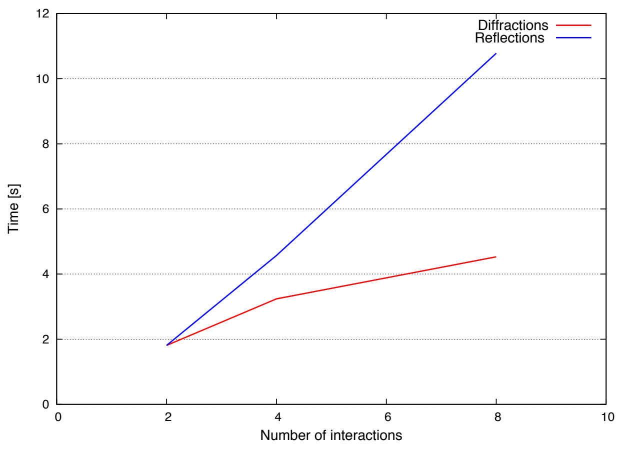

Fig. 9. Performance of our algorithm for varying levels of recursive interactions. Complexity grows approximately linearly.

transformed to a sum which can be expressed easily in matrix notation. The linear system is set up as:

$$C\frac{1}{d^{\gamma}}\prod_{i=1}^{N} f_i \rightarrow 1\log C - \gamma\log d + \sum_{i=1}^{N}\log f_i \tag{5}$$

We can than formulate the adaption of model parameters as a constrained least-squares problem in order to minimize the mean squared error between predicted and measured data:

$$\min_x ||F(x)||_2^2 = \min_x \sum_i F_i^2(x) \tag{6}$$

such that

$$F(x) = M \cdot x - d \tag{7}$$
$$A \cdot x \leq b \tag{8}$$
$$B \cdot x = c \tag{9}$$

Each row of the matrix M corresponds to one measurement location, whereas the columns are formed by the beams that reach the respective location, like travel distance of each arriving path and number of reflections and diffractions. The basic structure of M is depicted in Fig. 7 and is discussed in more details in the last paragraph of this section.

The vector d contains the measured path loss at each location, hence the optimal parameter vector $\hat{x}$ minimizes the mean squared error between the predicted and measured path loss with respect to the constraints (8) and additionally satisfying the equality constraints (9). Using the constraints we can incorporate expert knowledge on the propagation phenomena into the optimization problem, e.g., the path loss coefficient γ is known to be in the range between two for free space propagation and five inside densely populated cities. The optimal $\hat{x}$ can then be calculated by common solver algorithms like Gauss-Newton or Levenberg-Marquardt [34].

To the best of our knowledge, previous approaches to the adaption of model parameters from ray paths like the one of Mathar et. al [35] have been restricted to optimization of the strongest ray paths, only. First, they adapt their model parameters to best match their least attenuated ray path at each receiver location based on an initial parameter vector. Then they cyclically iterate between the computation of strongest ray paths and corresponding parameter estimation because changes in the model parameters can result in different strongest paths in the next iteration. They fail to provide a strict proof for convergence of their alternating approach, however they claim to achieve good results in practice after two or three iterations. Obviously, this procedure heavily depends on the initial parameter vector and is not guaranteed to find the global optimum because the optimization algorithm has no access to the information of all incoming ray paths, in a case with multiple strong propagation paths.

Our approach does not depend on an alternation between path computation and parameter estimation. We directly incorporate all paths from the beam hierarchy into the parameter estimation by binary encoding all existing paths. Hence, we can provide a closed form for the optimization algorithm and thereby inheriting all properties of the optimization procedure at hand. The method to incorporate the binary encoding in the optimization matrix M (cf. Fig. 7) is described below. For ease of understanding we describe the procedure only for the reflection effect. However, the concept is of course not limited to propagation paths based on reflection only but can easily be extended to support paths that are due to diffraction or other propagation phenomena.

Let R be the maximum recursion level of the reflection and N the maximum number of arriving paths. Each row of M is then of the form (without the leading constant)

$$\left(\log d_1 \quad \delta_{1,1} \ldots \delta_{1,R} \quad \ldots \quad \log d_N \quad \delta_{N,1} \ldots \delta_{N,R} \right) \tag{10}$$

where $\delta_{i,j}$ is a binary encoding such that

$$\delta_{i,j} = \begin{cases} 1 & \text{, if path } i \text{ has } j \text{ or more reflections} \\ 0 & \text{, otherwise.} \end{cases} \tag{11}$$

Hence, we use the $\delta_{i,j}$'s to switch certain parts of the matrix M on or off, so to speak. Thereby providing the required uniform input matrix for the optimization algorithms while still supporting a varying number of arriving paths for each receiver location.

With this method we have calculated the optimal parameter vectors for the widely known Munich dataset from the COST 231 project which contains a 3D model of Munich downtown and three measurement routes. Fig. 8 shows the shape of the optimal parameter vectors for each measurement path, separately. Although, the curves differ slightly in scale, they agree on the overall shape. The standard deviation between the optimal parameter vectors of route 1 and route 2 respectively, lie between 3 and 4 when compared to the optimal parameters of route 3. Therefore, we have chosen the parameter vector of the third measurement route for the actual computation of path loss. The adaption of model parameters to the third measurement route results in a very good agreement of prediction and measurement for the remaining measurement routes. When using those parameters, we achieve standard deviations between 6 dB and 8 dB for all three routes.

6 Evaluation

In this section we will take a look at the two most important properties of our algorithm. First we analyze the space and time complexity, and second we evaluate the accuracy, compared to real-world measurements and other works.

6.1 Complexity and Performance

The performance of the algorithm depends on several parameters. First of all the complexity of the scene geometry influences the beam splitting algorithm. It is vital to our algorithm that this visibility computation is done on the GPU, since it takes up most of the time in our algorithm. An important aspect of this is finding out the optimum configuration for the chosen platform. In our case, the dimensions of the computing grid on the CUDA device had to be optimized. Experiments showed that for this algorithm 128 threads are the optimal grid size on current devices.

However, the number of recursive reflection and diffraction steps also influences the complexity of the algorithm. In the beginning, we create only one beam in a certain direction, but with every split and reflection, it will spawn a number of additional beams, thus letting the number of beams grow exponentially in the worst case. However, beams tend to get very thin after a few reflections, so that on levels near the bottom of the beam tree there will be only a few children per beam.

Because of this, the size of the beam hierarchy becomes linear in the number of considered interactions, as can be seen in Fig. 9, and also depends on the spatial resolution of the framebuffer, which can be defined by the user in m/pixel. The evaluation itself is a simple polygon rasterization, and has been implemented using CUDA. We chose not to use OpenGL here, because it does not allow to render easily into a 3D texture, which is necessary for the delay spread histogram computation.

In the Munich scenario (see Fig. 1) there are approximately 80,000 vertices for the buildings. This is a reasonably complex scene, which is computed in several minutes on normal ray tracers for radio wave propagation (e.g. Schmitz et al. [21]). Our algorithm computes the scene in less than 3 seconds, even with complex recursive interactions. The machine used in this case was a Core2Duo with 2.4 GHz and a NVIDIA GeForce 8800 Ultra.

On the other hand the diffracted beams are usually much broader, often with an opening angle $> 90°$. This

Method	Accuracy	Time	Std. dev.	Delay spread	Reflections	Diffractions
Our method	5 m	1.8 s	6.18 dB	Yes	Yes	Yes
Wahl [36]	10 m	36 s	6.41 dB	No	Yes	Yes
Rick [28]	5 m	0.05 s	4.5 dB	No	No	Yes
Schmitz [21]	5 m	9m 20s	5.82 dB	Yes	Yes	Yes

TABLE 1
Comparison of our method to several other state of the art works. Our method supports all important propagation effects, and is apart from the work of Schmitz [21], the only method which simulates the delay spread.

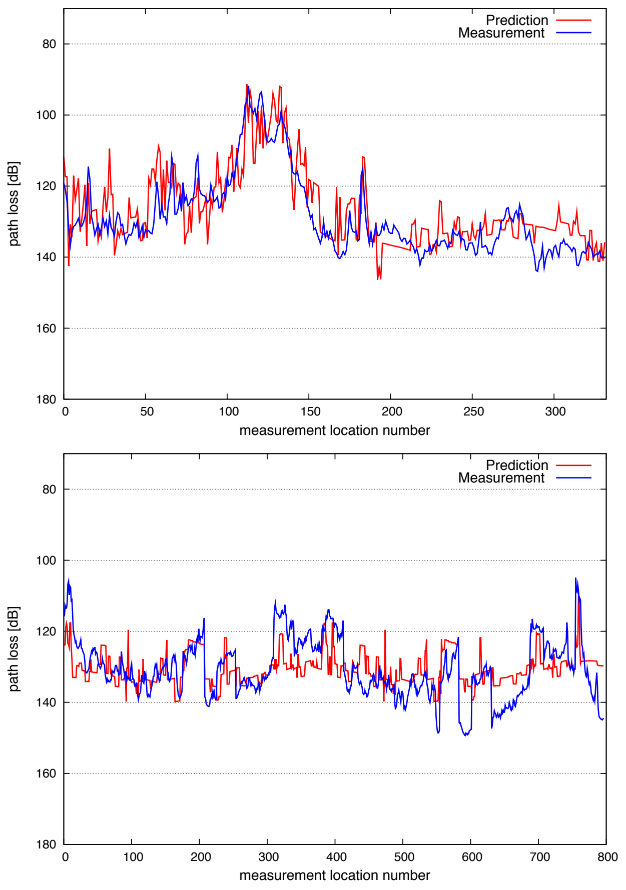

Fig. 10. A plot showing the measured results (blue) versus our simulation (red) for the downtown Munich scenario (top) and the Ilmenau scenario (bottom).

leads to much more spawned rays. However it is often not necessary to do more than one or two diffraction steps, because the signal strength is attenuated very fast by diffraction. Thus it is more important to create a propagation path that diffracts only once into a street, where reflection will carry on to propagate the signal.

6.2 Accuracy

For the evaluation with measured data, we use two different datasets. The first one is the above mentioned Munich dataset from the COST 231 project. The second is the dataset of the city of Ilmenau, as described by Schneider et al. [37]. This scenario contains measurements for three different base stations with highly accurate and detailed measurement samples. The actual measurement data had to be averaged and sampled down to our simulation grid's resolution, which is 5m. We chose this particular resolution, to be able to compare the Munich and Ilmenau scenarios. The former comes with measured data with 5m resolution. So using a higher resolution of the simulation grid provides no benefits. Hence we also use the same resolution on the Ilmenau data set. Memory and time requirements obviously grow proportionally to the grid resolution (i.e. linearly in the number of pixels).

In all our experiments we used the parameter optimization method described in Section 5. For any simulation method, a way of setting the simulation parameters needs to be used to provide sensible results. This can be done by educated guesses, or by our method, using sparse measurements and our optimization framework. Not using using either will lead to completely wrong results. The advantage of our method is that measurements lead to more correct results and the computed parameters are valid for different variations of a scenario. For all rasterization tasks we used the half-space rasterization algorithm, as described in Section 4.1.2. The recursion depth for the beam tracing was fixed to four reflections for all scenarios.

6.2.1 *Munich Scenario*

The measurements of this scenario can be taken as a reference solution for our algorithm and were taken on three different routes on street level. We compare the measured data with our simulation to estimate the quality of the simulation. It has to be noted that there are many more aspects to radio wave propagation than only interaction with static geometry. Especially moving objects, like people or cars, or even the weather can add significant, time based noise to measurements. This will influence measurements drastically, so that most other works leave these aspects out of the simulation, and use stochastic methods to model these effects on top of the ray or beam traced simulation.

In Fig. 10 we show one of the route plots in blue compared to our simulation in red. One can see that the overall shape of the curves match, which indicates that effects based on the scene geometry and its interaction with the radio waves are matched well by our algorithm. The higher level fluctuations in the measured signal most probably stem from dynamic effects which are not included in our algorithm. An example delay spread plot is shown in Fig. 11. Note the different spikes that are

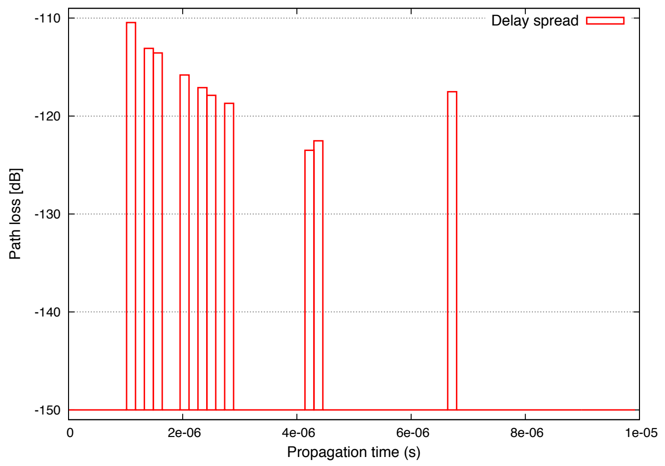

Fig. 11. A simulated delay spread histogram for one receiver position in the Munich scenario. The multiple peaks represent different propagations paths, which arise due to reflection and diffraction.

due to multiple reflections off of building walls. This will influence how well the receiver can decipher the transmitted signal, due to the resulting echoes.

The most common form of comparing radio wave propagation algorithms, is to compute the standard deviation of the errors between the simulation and the measurements. In Table 1 we compare our method to three other state of the art works. Besides the timings and the accuracy it is also most important to compare the features of the simulation. Although Rick [28] is able to run the Munich scenario at interactive rates, their method does not allow for the computation of reflections. Hence they are not able to compute a detailed delay spread, since this relies on multiple propagation paths, which are due to reflection and diffraction. It is noteworthy that Rick et al. get a better value for the standard deviation for this specific scenario. However, their approach has limitations. They are only able to predict the average signal strength, Multi-path propagation cannot be modeled. Optimally, one would use their approach to quickly optimize the antenna positions for best average signal strength, and then use our approach to optimize for delay spread characteristics. Wahl et al. [36] do not compute delay spread, although they state it might be incorporated into their approach. However this still has to be shown to be a viable option. The results from Schmitz and Kobbelt [21] are better, but with much longer computation times. Both the accuracy as well as the run time is due to the fact that they use a 3D ray tracer, which is computationally more expensive, but also models the given scenario in greater detail than our approach.

6.2.2 Ilmenau Scenario

This particular scenario was created and recorded by the TU Ilmenau [37] and contains about 380,000 vertices, see Fig. 12. The measurements contain data for three different base stations and were sampled very densely. Because of the fact that we used only a resolution of five meters for our simulation, the measurement data had to be sampled down. For example there exist more than 170,000 sample points for the first base station. We reduced this amount of data to less than 900 samples by averaging to fit our simulation resolution.

Base Station	Std. dev.	Measured samples
BS1	6.9 dB	842
BS2-1	6.2 dB	403
BS2-2	4.7 dB	98

TABLE 2
Results for the city of Ilmenau scenario.

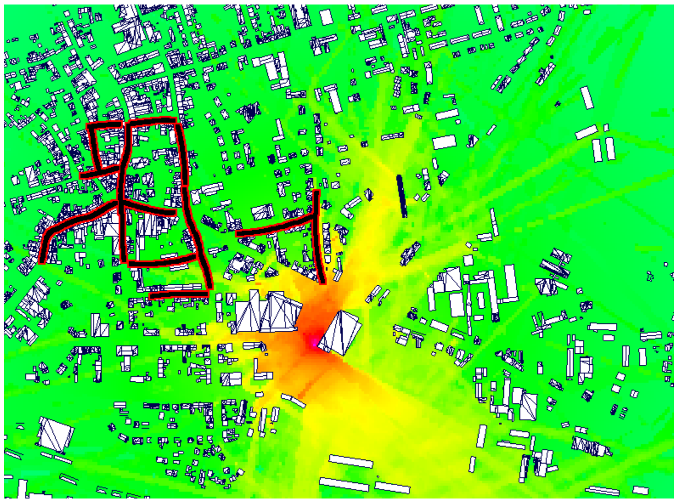

Fig. 12. The city of Ilmenau scenario, with the propagation result for base station 1, and the measurement routes in black.

The geometry of the city already contains some slopes towards the border of the city model. This can lead to wrong predictions, since our simulation model so far assumes relatively flat urban scenarios. This leads to a less accurate prediction than in the Munich setting, but still comparable to the earlier presented results from our and other works. The simulation results can be seen in Tab. 2 and Fig. 10, and a visual result in Fig. 12. Although the simulation data in Fig. 10 matches the measured data quite well, one can see that our approach overestimates the signal strength at some points. This might be due to landscape geometry, which we neglect in our 2D approach, or other factors that are not reflected in our model. Nevertheless we achieve a standard deviation between 4.7 and 6.9 dB for the simulation results compared to the measurements.

7 Conclusion

We have shown a fast and accurate algorithm that can accurately simulate radio wave propagation in urban environments, including important effects like multipath propagation due to reflection and diffraction, and predict the resulting delay spread.

As future work, we would like to implement a full 3D beam tracer. This leads to a number of interesting, but difficult questions, but will also solve several restrictions. Currently, scenes with hills and valleys will not be modeled very accurately. Our 2D method will predict line of sight paths or wall-reflection paths that do not exist in reality. This already becomes somewhat apparent in the Ilmenau scenario. Hence extension to 3D is necessary for the method to be more general.

However the beam-splitting by rasterization as used in the presented work will not work anymore when using a true 3D approach. Also beams and beam-hierarchies can have non-trivial geometry. Especially in the case of slightly curved surfaces one has to take care that the reflected EM-field modeled by the beam-hierarchy is still continuous. In 2D this is more easy than in 3D, where the edges of a reflected beam don't have to intersect in one point anymore.

8 Acknowledgments

The authors would like to thank Christian Schneider of the TU Ilmenau for supplying measurement data and the corresponding geometric city model.

References

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

[2] A. Lehnert, "Systematic errors of the ray-tracing algorithm," *Applied Acoustics*, no. 38, pp. 207–221, 1993.

[3] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*. New York, NY, USA: ACM, 2008, pp. 1–15.

[4] R. Overbeck, R. Ramamoorthi, and W. R. Mark, "A real-time beam tracer with application to exact soft shadows," in *EGSR*, 2007.

[5] A. Schmitz, T. Rick, T. Karolski, T. Kuhlen, and L. Kobbelt, "Simulation of Radio Wave Propagation by Beam Tracing," K. Debattista, D. Weiskopf, and J. Comba, Eds. Munich, Germany: Eurographics Association, 2009, pp. 17–24. [Online]. Available: http://www.eg.org/EG/DL/WS/EGPGV/EGPGV09/017-024.pdf

[6] T. Whitted, "An improved illumination model for shaded display," *Commun. ACM*, vol. 23, no. 6, pp. 343–349, 1980.

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

[8] N. A. Carr, J. Hoberock, K. Crane, and J. C. Hart, "Fast gpu ray tracing of dynamic meshes using geometry images," in *GI '06: Proceedings of Graphics Interface 2006*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2006, pp. 203–209.

[9] C. Dachsbacher, M. Stamminger, G. Drettakis, and F. Durand, "Implicit visibility and antiradiance for interactive global illumination," in *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*. New York, NY, USA: ACM, 2007, p. 61.

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

[11] N. Tsingos, E. Gallo, and G. Drettakis, "Perceptual audio rendering of complex virtual environments," in *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*. New York, NY, USA: ACM, 2004, pp. 249–258.

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

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

[14] T. S. Rappaport, *Wireless Communications: Principles and Practice*. Prentice-Hall, Inc., 1995.

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

[16] M. Hata, "Empirical formula for propagation loss in land mobile radio services," *IEEE Transactions on Vehicular Technology*, vol. 29, no. 3, pp. 317–325, Aug 1980.

[17] F. Ikegami, S. Yoshida, T. Takeuchi, and M. Umehira, "Propagation factors controlling mean field strength on urban streets," *IEEE Transactions on Antennas and Propagation*, vol. 32, pp. 822–829, 1984.

[18] 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.

[19] F. Ikegami, T. Takeuchi, and S. Yoshida, "Theoretical prediction of mean field strength for urban mobile radio," *IEEE Transactions on Antennas and Propagation*, vol. 39, pp. 299–302, 1991.

[20] K. R. Schaubach, N. J. D. IV, and T. S. Rappaport, "A ray tracing method for predicting path loss and delay spread in microcellular environments," in *Proc. IEEE Vehicular Technology Conference*, vol. 2, May 1992, pp. 932–935.

[21] A. Schmitz and L. Kobbelt, "Wave propagation using the photon path map," in *PE-WASUN '06*. New York, NY, USA: ACM, 2006, pp. 158–161.

[22] S.-C. Kim, B. J. Guarino, T. M. W. III, V. Erceg, S. J. Fortune, R. A. Valenzuela, L. W. Thomas, J. Ling, and J. D. Moore, "Radio propagation measurements and prediction using three-dimensional ray tracing in urban environments at 908 MHz and 1.9 GHz," *IEEE Trans. Veh. Technol.*, vol. 48, pp. 931–946, 1999.

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

[24] A. Chandak, C. Lauterbach, M. Taylor, Z. Ren, and D. Manocha, "Ad-frustum: Adaptive frustum tracing for interactive sound propagation," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 6, pp. 1707–1722, Nov.-Dec. 2008.

[25] A. Rajkumar, B. F. Naylor, F. Feisullin, and L. Rogers, "Predicting rf coverage in large environments using ray-beam tracing and partitioning tree represented geometry," *Wirel. Netw.*, vol. 2, no. 2, pp. 143–154, 1996.

[26] S. Fortune, "A beam-tracing algorithm for prediction of indoor radio propagation," in *WACG*, ser. Lecture Notes in Computer Science, M. C. Lin and D. Manocha, Eds., vol. 1148. Springer, 1996, pp. 157–166.

[27] D. Catrein, M. Reyer, and T. Rick, "Accelerating radio wave propagation predictions by implementation on graphics hardware," in *Proc. IEEE Vehicular Technology Conference*, Dublin, Ireland, 2007, pp. 510–514.

[28] T. Rick and R. Mathar, "Fast edge-diffraction-based radio wave propagation model for graphics hardware," in *Proc. IEEE 2nd International ITG Conference on Antennas*, Munich, Germany, March 2007, pp. 15–19.

[29] J. Pineda, "A parallel algorithm for polygon rasterization," *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 17–20, 1988.

[30] A. S. Glassner, *Graphics Gems*. Orlando, FL, USA: Academic Press, Inc., 1990.

[31] *NVIDIA CUDA Reference Manual 2.2*, 2009.

[32] A. Munshi, Ed., *The OpenCL Specification*. Khronos OpenCL Working Group, 2009.

[33] J. B. Keller, "Geometrical theory of diffraction," *Journal of the Optical Society of America*, vol. 52, pp. 116–130, 1962.

[34] K. Levenberg, "A method for the solution of certain problems in least squares," *Appl. Math.*, vol. 2, pp. 164–168, 1944.

[35] R. Mathar, M. Reyer, and M. Schmeink, "3D ray launching algorithm for urban field strength prediction," in *Proceedings: ICC*, 2007.

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

[37] C. Schneider, "Multi-user mimo channel reference data for channel modelling and system evaluation from measurements," in *Int. ITG Workshop on Smart Antennas*. EURASIP, February 2009.

Arne Schmitz received his Dipl.-Inform. degree in computer science from RWTH Aachen University, Germany in 2005, where he is currently pursuing his Ph.D. degree at the Computer Graphics Groupe. His main research interests are in radio wave propagation, global illumination and rendering.

Tobias Rick received his Dipl.-Inform. degree in computer science from RWTH Aachen University, Germany in 2006, where he is currently working towards the Ph.D. degree. His main research interests are in radio wave propagation and scientific computing.

Thomas Karolski is currently an undergraduate student helper at the Computer Graphics Group of the RWTH Aachen University.

Torsten Kuhlen is head of the Virtual Reality Group at RWTH Aachen University. In 2008, he was appointed to a professorship in the Department of Computer Science. His research interests include basic technologies as well as scientific applications of VR. He is the speaker of the steering committee of the VR/AR chapter of Germany's computer society.

Leif Kobbelt is a full professor and the head of the Computer Graphics Group at the RWTH Aachen University, Germany. His research interests include all areas of Computer Graphics and Geometry Processing with a focus on multiresolution and free-form modeling as well as the efficient handling of polygonal mesh data. He was a senior researcher at the Max-Planck-Institute for Computer Sciences in Saarbrücken, Germany from 1999 to 2000 and received his Habilitation degree from the University of Erlangen, Germany where he worked from 1996 to 1999. In 1995/96 he spent a post-doctorate year at the University of Wisconsin, Madison. He received his masters degree in (1992) and Ph.D. in (1994) from the University of Karlsruhe, Germany. Over the last few years he has authored many research papers in top journals and conferences and served on several program committees.