The University Of Sheffield.

The Study of a Universal High-performance Radio Propagation Model for 3D Coverage Prediction of Dense Network in Indoor and Outdoor Environment

Hanye Hu

This thesis is submitted for the approval of the

Doctor of Philosophy

The University of Sheffield

Department of Electronic and Electrical Engineering

February 2020

Abstract

Radio propagation models are used to facilitate the planning of wireless networks. Over the decades, with the evolvement from 2G to 5G technology and the use of higher frequency bands, there is an emerging demand for more accuracy from the propagation model. Although it is well acknowledged that deterministic models are more desirable for meeting the requirement, there still lacks an efficient solution, especially when the usage is for the full 3D coverage prediction which is necessary for the 5G heterogeneous network (HetNet) planning.

This study starts with the investigation of current propagation models and their principles in the literature to understand the challenges in providing an accurate and efficient solution for the radio prediction in 3D. Then two propagation models with desired speeds using different techniques are compared side by side to study the method that is more suitable for the problem of interest. It is learnt that the ray-launching method should be the better candidate for its good balance in efficiency and accuracy.

However, in the literature, the existing ray-launching methods are still not presenting speed fast enough for 5G network planning, due to the high density of cells and the complexity of the HetNet environment. Some adoptions of hardware acceleration are showing potential, but the presented speed boost is still much less than the advertised improvement from hardware manufacturers. Therefore, a prototyping ray-launching model with hardware acceleration is firstly implemented in this research, to study the possibility of using hardware acceleration for providing an efficient solution. The prototyping model does demonstrate a considerable performance boost by using suitable acceleration methods,

but the final speed is still not enough for the need. It is learnt that the core limitation for providing the desired solution is from the algorithm complexity of 3D ray-launching.

Therefore, another propagation model based on tube-launching is created to reduce the algorithm complexity. To overcome the known challenges in tube-launching, e.g. tube intersection test and tube chopping, an innovative ray splitting method is proposed to approximate the tube shape and adapt the calculation resolution. Then, the implementation and validation of this model are presented in different indoor and outdoor scenarios. The results prove that the tube-launching propagation model does provide the desired accuracy, efficiency and versatility for 5G network planning.

In the end, benefiting from the adaptive characteristic, the above model is further extended to a multi-resolution propagation model for providing outdoor-to-indoor (O2I) and indoor-to-outdoor (I2O) calculation. Based on the mechanism, the proposed model provides a seamless transition across the boundary between the outdoor and indoor environments using a deterministic approach, while the existing models in the literature use non-deterministic transitions with re-launching behaviours on the boundary. In an O2I scenario, the multi-resolution model is shown with efficient simulation and a good match with the measurement.

List of Figures

List of Tables

List of Acronyms

AABB	axis-aligned bounding boxes
ALU	algorithm logic units
AoA	angle of arrival
AoD	angle of departure
BVH	bounding volume hierarchy
CIR	channel impulse response
CORLA	cube-oriented ray launching algorithm
CPU	central processing unit
CUDA	Compute Unified Device Architecture
DFS	depth-first search
DSP	digital signal processor
EM	electromagnetic
FDPF	frequency domain parflow
FDTD	finite-difference time-domain
FPGA	field-programmable gate arrays
GA	genetic algorithm
GBSM	geometry-based stochastic models
GIS	geographic information system
GO	geometrical optics
GORL	geometrical optics based ray launching
GPU	graphics processing unit
HetNet	heterogeneous networks
HPC	high performance computing
IRLA	intelligent ray launching algorithm
mmWave	millimetre wave
MR-TLARS	multi-resolution tube-launching adaptive ray splitting
NGSM	non-geometrical stochastic models
OpenCL	Open Computing Language
PDF	probability distribution function
PDP	power delay profile
PO	physical optics
PoI	point-of-interest
PSO	particle swarm optimization
RMSE	root mean square error
SA	Simulated Annealing
SAH	surface area heuristic
SUI	Stanford University Interim
TE	transverse electric
TLM	transmission-line matrix

TM	transverse magnetic
UDN	ultra-dense network
UML	Unified Modelling Language
UTD	uniform theory of diffraction

Table of Contents

1 Introduction

1.1 Research Background

Radio propagation models are widely used to predicate the strength of electromagnetic (EM) field within a modelled environment. Depending on the scopes, the result can be used to understand the characteristics of EM components or can be used to pre-evaluate the wireless link budget between the transmitter and a point-of-interest (PoI). Due to the limitation of computational capability, a good trade-off between the scopes of the environment and the complexity of the model algorithms has to be investigated carefully. Ideally, the methods solving Maxwell's equations can be used for an accurate estimation of the wave phase, polarization and attenuation. However, a detailed modelling of the environment and a large amount of computational burden are needed for small scopes [1], e.g. for tens of meters. Thus, such an approach is not feasible for wireless network planning, which could need the evaluation of radio links between a few kilometres. To cope with the scale issue, empirical models based on analytical study and measurement of typical scenarios are developed to provide adequate estimation based on limited environment modelling for communication channels over long distances [2].

While for the medium-range from a few hundred meters to a few kilometres, a trade-off approach is promising, which provides an estimation more accurate than the empirical model while still needs less computing time and environment information than Maxwell's equations solver. The deployment of heterogeneous networks (HetNet) has brought more attention to such an approach, as the network quality within a few hundred meters is becoming the major concern for small cells. With the introduction of 5G, especially the use of millimetre wave (mmWave), which is very sensitive to the

surrounding environment [3], the medium scope radio model is becoming necessary for pre-evaluating the network plan before deployment.

The deterministic models provide accurate options for addressing the medium scope radio prediction. Some examples are the ray-tracing, ray-launching and differential equation solver like finite-difference time-domain (FDTD) method [4]. The ray-tracing algorithms predict EM waves propagations with given positions of both transmitters and the receivers. The rays satisfying geometrical optics (GO) rules for the given positions are generated by various techniques. As for the ray-launching algorithms, the rays are not traced specifically for exact receiver positions. Instead, the rays are propagated from a transmitter and all the reflections, diffractions, and scattering that happened during the transmission are counted. In some literature, the ray-tracing is referred to as the image method, and the ray-launching is referred to as the brute force method due to their ray generation strategies.

These methods can achieve satisfactory accuracy close to the measurement [1]. However, the calculation burden could still be significant for scenarios with a radius of thousand meters at high resolution. Based on the basic concept, derived forms are proposed to improve the calculation efficiency or simplify the modelling approach. Some methods use a hybrid approach to offload part of the complex calculation to the empirical methods. Some methods reduce the accounted dimension from 3D to 2.5D/2D. Specifically for the ray-based approaches, some methods use different shapes to model the wave, e.g. triangle tube and square tube [5].

However, the above methods are not suitable for the pre-evaluation of the network plan. On the one hand, to evaluate the network, a full 3D volume radio prediction within

the range of interest is needed including the vertical variation. The demand for result along the *z* dimension comes from the need for indoor coverage provided from outdoor base stations. Many ray-based methods, especially ray-tracing, focus on the result in a limited set of points, thus their computational burden is much less than focusing on all the points in the considered 3D environment. Then the performance would be less a concern in such case. On the other hand, a method may not scale well for the calculation of many transmitters. One of the key trends of 5G deployment is the dense small cell, e.g. tens of small cells could be placed within a square kilometre [6]. This means the time consumption of one propagation model would be increased by tens of times to provide the evaluation of a small area. Not to mention the evaluation needs to be repeated a few times in a trial-and-error process to find out a better network plan. Thus, there is a demand for a propagation model providing high speed full 3D volume calculation.

1.2 Statement of the Problem

The full 3D volume calculation should provide radio prediction at each location within a range of interest. The calculation along *z* dimension would have a significant impact on the degree of complexity. Depending on the ratio of the *z* range over the resolution, the complexity could be increased by tens of times for the methods that put the focus on the street level or even a few points. This is especially the case for the ray-tracing method which is targeting at point-to-point. Based on the number of ray-object interactions counted in a method, the ray-tracing complexity increases exponentially while the ray-launching complexity increases linearly [7].

On the other hand, ray-launching approaches are discrete naturally. Thus, the idea of collection sphere [8] is developed to sample the discrete rays into a continuous result within a given resolution. One example of the collection spheres can be referred to as a 3D grid which is composed of uniform collection voxels. Therefore, full estimated coverage can be provided as long as the ray-launching method can guarantee that all the voxels can be reached.

However, there is a chance of double-counting the rays especially for the collection voxel close to the transmitter, where the rays are dense and could end up going through the same voxel [8]. Also, for the rays reflected on the same polygon from the transmitter, double-counting should be avoided. The handling of the issue becomes complex after more interactions.

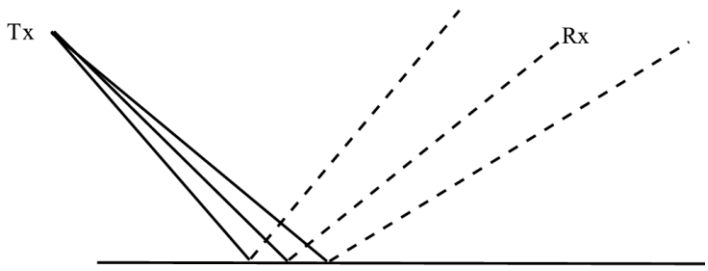


Figure 1-1 Angular dispersion increases the gap between rays after reflection

Another issue arises as the separation between neighbour rays could continuously increase due to reflection, since the ray travelling distance is extended, as shown in Figure 1-1. Unavoidably, the separation could become larger than the collection voxel and lead to gaps in the sampled result. Without the consideration of angular dispersion, a method would not be suitable for the given 3D volume calculation. Angular dispersion is the main challenge of the studied problem, as it represents the dependency of complexity on the number of interactions. Neglecting the problem would decouple the dependency.

To solve the practical angular dispersion problem, the tube-tracing method is developed to model the continuous wave by tubes with volumes instead of discrete rays [9, 10]. However, this approach brings up the new challenges of tube splitting and hit detection. Firstly, on hitting an object partially, an incident tube needs to be split into a few sub tubes, among which one will be the actual tube covering the full hit, and the rest sub tubes will be free from the hit. Secondly, as the tube propagates further or bounces back from a reflection, the tube size could become too large that an object can be fully contained within a tube, so that the hit detection may be missed.

Another approach solving the angular dispersion problem is to use the FDTD method, which does not have the problem intrinsically, as it does not model the rays but instead solves the time-dependent Maxwell's equations in a discrete grid. However, the FDTD method requires the grid sampling of the scenario in advance, so it is raster-based instead of vector-based. The raster model has its limitation of resolving fine details in the close distance, e.g. it is hard to represent two parallel walls with 0.5m distance given a 1m raster resolution. Also, the edge of windows and doors cannot be accurately represented.

Therefore, an effective solution to the problems arising in the 3D volume calculation would face enormous challenges from modelling and computational complexity. Specifically, the goal of such a radio propagation model should be able to calculate one transmitter for a coverage radius of 1 kilometre in a dense urban area within minutes. Such a requirement is necessary based on state-of-the-art propagation models' performance [11-13], and the fact that tens of small cells need to be planned in such area.

Meanwhile, for the planning of HetNet, the support for O2I and I2O calculation is necessary as well. Fundamentally, the challenge of O2I is due to that different levels of

scenario detail are provided for the outdoor and indoor areas. Therefore, their desired calculation resolutions are also different. A solution providing an efficient transition along the boundary between the outdoor and indoor scenarios is needed.

This research devotes itself to develop a vector-based high-performance 3D radio propagation model providing the desired performance while addressing the arising problems discussed above like angular dispersion and double-counting. The limitation of existing methods and their potential in providing the 3D volume calculation are studied. If the limitation is due to the lack of computing power, it is reasonable to re-exam the ideal solution with an up-to-date computing device and hardware acceleration. Then a proposed adaptive tube-launching method is implemented. In the end, the performance and accuracy of the model are evaluated.

1.3 Significance of the Study

One of the key features for 5G is the introduction of the ultra-dense network (UDN), e.g. small cells with a coverage radius around 100m will be massively and densely deployed to meet the explosive demand of data traffic [14]. Careful planning has to be made to determine the exact locations for the massive number of the small cells, to minimize their interference and maximize their performance thus efficiency. The CAPEX for the full roll-out of 5G in Britain is estimated at £12 billion in [15]. Therefore, any savings in the network deployment will be significantly magnified by the total volume. Meanwhile, due to the use of mmWave, the 5G coverage would be very sensitive to the surrounding environment [3].

Thus, it can be recognized that there is a strong demand for a capable radio propagation model to help the design, planning and evaluation of dense small cell networks. The propagation model should have enough accuracy to provide recognizable improvement for the network planning over traditional experience-guided approaches or other existing radio models. For reference, the cost of a 3G network could be reduced by about 20%-40% with simulation assisted planning [16].

Yet as discussed in Section 1.2 and Section 2.2, the challenges of a medium-range radio model are not well addressed by existing methods. Firstly, some of the methods miss the consideration of problems like angular dispersion. Secondly, some methods have made simplification as compensation to the computing efficiency, thus the scarification of the accuracy. Lastly, some methods are trying to solve the challenges by innovating the modelling method, like tube-launching, but new challenges, which aren't practically addressed, arise at the same time.

Also, for the application in O2I and I2O scenarios, many solutions are still based on empirical methods [17-20], due to the mixed complexity from both scenarios. While for the other proposed O2I deterministic models, they are composed of hybrid approaches that introduce a discontinued transition along the boundary between the outdoor and indoor scenarios. Such a transition could lose the respect of the actual on-site structure, as discussed in Section 2.2.3.

Thus, there is a gap in the research area providing a practical deterministic radio model matching the industrial demand and addressing the known computing challenges. The goal of this study is to develop a feasible solution to the problem while maintaining

competitive efficiency and accuracy. So, the industry could be provided with a modern radio propagation model helping and improving network planning.

1.4 Scope and Delimitation

Firstly, the study aims to the medium-range radio prediction, e.g. from a few hundred meters to a few kilometres. This range bears the improved environment model detail over the long haul. Therefore, the variation of the environment in small scale matters especially for the mmWave used in 5G which is sensitive in providing the coverage. On the other hand, the complexity raised from 3D volume calculation needs to be handled with respect to the detail. Also, the cost of the geographic information system (GIS) data rises with its resolution. Therefore, a balance in the required model information, computing efficiency and accuracy is expected for the medium range.

Secondly, the study aims to provide a deterministic radio model, instead of an empirical model, which generalizes the statistics from measurements in different typical environments, resulting in being sensitive to the change of scenarios [5]. Such dependency problem is enlarged by the improved environment details and the consideration of variation in small scopes. On the other hand, the deterministic method provides a universal approach for the calculation in different types of scenarios. Thus, a significant advantage is provided in saving the time needed for tuning an empirical model for scenarios. Not to mention the deterministic model can provide satisfactory accuracy close to the measurement [1].

Thirdly, the focus of the study is to improve the efficiency of a deterministic method. The wireless channel modelling has been a hot research topic over the decades. There exist mature theories for modelling the radio using empirical, deterministic and

hybrid methods. The identified gap is in applying the methods in 3D volume calculation with the significant rising of complexity. Especially the calculation speed is becoming critical for efficient estimation and planning of a dense small cell network. There are methods reducing the complexity by offloading the calculation along the z dimension to empirical approaches [21] or simply perform a transition from 3D to 2.5D/2D [22]. However, these simplifications would inevitably lose certain shadowing and visibility information, especially for the mmWave which is approximating the behaviour of light. Thus, this study aims to persist a deterministic based approach in full 3D volume.

Lastly, the proposed method in this study would be based on narrow band. Ideally, the 5G could use bandwidth beyond a hundred megahertz, for which there will be the frequency-selective effect. However, ray-based methods are having difficulty providing wideband calculation [1]. To provide the wideband consideration, one approach is to perform multiple runs of the same scenario at different frequencies, i.e. sample the wideband into narrowband. Also, an extension on a ray-based method with an empirical approach could be used to approximate the frequency selective effect. After all, the focus of this study will be based on the ray method for narrowband.

1.5 Structure of the Thesis

The following Chapter 2 will give the literature review for the up-to-date propagation models. Both the empirical and deterministic models will be introduced. Although the main focus of this study is deterministic model, stochastic method could be useful for compensating and extending the limit of a deterministic method. In the meantime, the principles for the ray-based methods will be given as well.

Then Chapter 3 will focus on the study of a preferred method for the aimed 3D volume calculation among the deterministic candidates. Firstly, the complexity of ray-tracing and ray-launching methods will be estimated for the problem domain. Then a ray-launching model and a transmission-line matrix (TLM) like model, which are both presenting high calculation speed, are integrated into a building modelling tool to provide a side-by-side comparison of their efficiency and accuracy.

Thereafter, Chapter 4 will start the prototyping of a GO-based ray-launching engine powered by hardware acceleration. The goal is to investigate the feasibility of using the hardware acceleration and modern processors to overcome the raised complexity from the 3D volume calculation. The actual computing complexity, bottleneck and other challenges will be understood through the implementation and simulation of the prototyping model.

Based on the prior experience and conclusion, Chapter 5 will propose a tube-launching method based on central ray splitting, which aims to address the complexity problem identified in the prototyping model. Implementation of the method with improved module designs will be provided to evaluate the performance. Validation against measurements in different scenarios will be given in the end, to prove the accuracy and versatility of the model.

Then in Chapter 6, the extension of the tube-launching method will be given to support the O2I and I2O calculation based on a deterministic approach. The implementation and simulation of the model will be given as well to demonstrate the efficiency and capability of the proposed method in hybrid environments.

In the end, the summary of this study and the identified future work will be concluded in Chapter 0.

2 Review of Related Literature

This chapter introduces the state-of-the-art radio propagation prediction methods. In general, existing propagation methods applied to analyse communication qualities, and simulate the performance of networks can be categorized into two kinds, i.e. the empirical models and the deterministic models [6]. The deterministic models generate relative pathloss and channel impulse response (CIR) in high accuracy, but the computational complexity is relatively high. The empirical ones are targeting to find a good trade-off between the accuracy and computational burden. Therefore, in this chapter, the empirical radio propagation models are firstly introduced, followed by the deterministic radio propagation models. The discussion will be concluded at the end of the chapter.

2.1 Empirical Radio Propagation Models

In this section, the concepts and problems of empirical radio propagation models are introduced. The empirical models can also be categorized into stochastic models, which characterise the scatterers in the channel by certain types of stochastic distribution, and map-based models, which approximate the actual environment by simplified structures. The stochastic models are applicable to simulate the fast time-varying channels, because of their good trade-off between accuracy and computational burden [23], while the map-based models provide efficient pathloss calculation over various ranges and scenarios.

2.1.1 Stochastic Models

The stochastic models do not explicitly model the structure of the environment, e.g. buildings and walls, but assume the distribution of scatterers around the transmitter and receiver. Depending on the geometrical representation of the scatterers, the stochastic

models can be further separated into geometry-based stochastic models (GBSM) and non-geometrical stochastic models (NGSM).

2.1.1.1 Geometry-based Stochastic Model

The GBSM modelled the CIR based on the geometrical location of scatterers. The channel model applies the fundamental laws of specular reflection, diffraction, and scattering of EM waves for the positions of the scatterers to compute the CIR, from which the statistical properties of the channel can be obtained, such as the angle of arrival (AoA), angle of departure (AoD) and time delay etc.

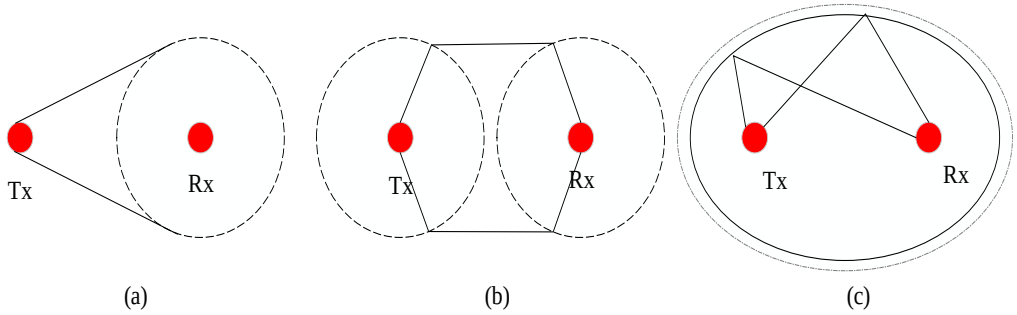


Figure 2-1 The three typical GBSMs: (a) one-ring model; (b) two-ring model; (c) elliptical model

The often-used GBSMs include the one-ring, two-ring, and elliptical scattering models. For some 3D scenarios, the cylinder scattering model has also been applied [24]. The one-ring model is suitable for describing the environment in which the receiver is surrounded by some scatterers. The two-ring model assumes that the scatterers exist surrounding both the transmitter and the receiver. Therefore, the transmitter and the receiver will be set in the middle of the ring. The one-ring and two-ring models are frequency nonselective, so these two models are often used in the narrowband system [25]. On the other hand, the elliptical model is used to model the wideband channel. The

elliptical models place scatterers on a set of ellipses with varying focal lengths whose foci correspond to the transmitter's and the receiver's positions [26]. Typical GBSMs are shown in Figure 2-1. For 3D application, the cylinder scattering model is used, in which one of the transmitters and receivers will be set with the elevation angle and height value. The sketch of the cylinder scattering model is shown in Figure 2-2. At the scatterer placement step, these typical shaped scattering models are called regular-shaped GBSM. In contrast, there are irregular-shaped GBSM which provides better adaptivity to the environment with the cost of a more complex setup [27, 28].

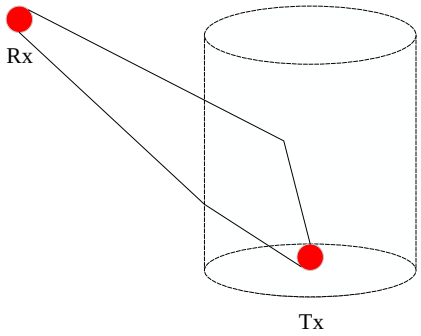


Figure 2-2 The 3D cylinder GBSM.

2.1.1.2 Non-geometrical Stochastic Models

In terms of the NGSMs, they do not assume the physical propagation environment, but can still describe physical parameters, such as AoD, AoA, and delay stochastically by following PDF of the scatterers' properties. Comparing GBSM and NGSM, NGSM uses measurement to get the statistical properties of the parameter in the channel model, while GBSM can use measurement or reference model to get the parameters in the channel [26]. In [29], an NGSM for capturing indoor channel CIR is proposed. Later, the authors extended the method to capture the MIMO channel parameters [30]. The NGSM is also applied in some standard channel models, for example, the IEEE 802.11 TGn model [31].

As concluded in [32], the NGSM is challenging to access the accurate channel parameter when ignoring all the physical environment characteristics, but it can easily capture the rapid blockage effect by mobile vehicles or other obstacles of the wireless channel, due to that the appearance of these blockages is dynamic rather than static in the scenario.

2.1.2 Map-based Empirical Models

The map-based empirical models do partially consider the on-site environment by simplified forms. Therefore, they provide better adaptiveness to the actual scenario than the stochastic models, while still maintain good efficiency. In outdoor environments, the knife-edge model is widely adopted to provide the estimation of the shadowing effect after diffraction. The fundamental concept is to approximate a diffraction obstacle, e.g. a hill or a tall building, as a knife-edge shape. The single knife-edge model is proposed in [33]. Then the derived forms with double edges [34] and multiple edges [35] are proposed to adapt scenarios in urban areas with many buildings. Also, different shapes with closer approximation to obstacles are used, but they are not widely adopted due to the complexity [36]. The knife-edge methods have high efficiency in providing the result for the complex diffraction problem and are proved with good accuracy over long haul [37, 38]. Standard channel models [39] and hybrid models [21, 40] have adopted the method to simulate the effect of rooftop diffraction. Some deterministic models also use the knife-edge method to reduce the calculation complexity of diffraction [21].

However, as an empirical method, the knife-edge has its limitation in scenarios with more environmental details. The approximation of knife-edge shape is less accurate as the range becomes shorter, since the relative scale of building shapes increases [36]. On the

other hand, although the knife-edge provides the result on z dimension, it has the limitation in mixed environments with O2I transition.

In indoor environment, another widely used approach is the multi-wall model, which accounts the penetration through walls explicitly. A frequently used example is the COST231 multi-wall model [41], for which the pathloss formula is given as

$$PL = PL_0 + 10nlog(d) + \sum_{i=1}^{N} PL_i, \quad (2\text{-}1)$$

where PL_0 is the reference pathloss measured at a distance d_0, n is the pathloss exponent factor which is 2 for 1.9GHz, d is the distance between the transmitter and receiver and PL_i is the loss factor for the ith wall. In [42], the basic multi-wall concept is extended to consider the floor specifically to form a multi-wall-and-floor model. Other variations work on more explicit expression of the PL_i factor, for example the different wall thicknesses with the same material is considered in [43].

A close accuracy between the multi-wall model and ray-tracing model is presented with the measurement at 865MHz in [44]. However, the multi-wall approach does not consider the effect of diffraction and reflection, it is less applicable for higher frequencies especially the mmWave which would incur severe penetration loss.

2.1.3 Standard Channel Models

Based on the fundamental modelling mechanisms introduced above, standardized models are developed with measurement tuning at different frequencies, ranges and scenarios by research projects and associations. They act as general guidance for wireless network planning. These models can be classified into the ones without and with the expression of physical structure between the transmitter and receiver. For the prior category, the examples are the Stanford University Interim (SUI) model [45], the COST231

Hata model [41], the HATA-OKUMURA model [46] and the ECC-33 model [47]. In general, their formulas are composed by a set of terms reflecting the variation in range, frequency, transmitter antenna height and receiver antenna height. The composition and factors of the formula could change for different scenario types, e.g. urban or rural, as a rough expression of the environment. In addition, the SUI model contains the categorization of terrain.

Since no physical structure is expressed in these formulas, they provide high-speed calculation. As a result, these models can always provide a result for every location in the scenario, which may not be valid. In [48], they are validated against drive tests in different environments, while in urban areas, the RMSEs are shown in the range from 10dB to 20dB. Also, their MEs are shown to be above 10dB from measurement for fixed wireless access [49]. From the result comparison plot in [48], the major difference happens from 200m to 1000m. This may be because that, before 200m, the channel condition is close to LOS, and after 1000m, the impact of environmental variation is less significant, while in between, the on-site structures dominate the signal condition.

To adapt the mitigation to 4G and 5G, the recently published standard models are introducing more explicit expression for the physical structures to form a hybrid approach with the stochastic method and map-based method. The examples are the METIS model [20], the 3GPP channel model (3GPP TR38.901) [50] and the IEEE 802.11ay model [51].

METIS is one of the first models proposing hybrid approaches for channel estimation. In the map-based part, it uses simplified shapes, e.g. cuboids, to approximate the physical structures in categorized scenario types. In addition to the general types, the stadium and open-air festival are specified, for which geometrical structures are irregular

compared to the urban or indoor scenarios. Then the ray-tracing method with the consideration of diffraction, specular reflection, diffuse scattering and blocking is used on the established scenario to compute the pathloss and shadowing (large-scale). Specifically, the diffraction is simulated using the knife-edge method, while the fading and other calculations (small-scale) are computed by the stochastic model. This hybrid approach is shown with a good trade-off between accuracy and computational burden [6]. The 3GPP hybrid model adopts a similar approach from the METIS, while it introduces the modelling of atmospheric absorption at 53-67GHz. But the 3GPP parameterizes fewer scenario types, which are urban macro cell, rural macro cell, indoor office and street canyon.

The IEEE 802.11ay model embeds both the map-based and stochastic methods into a ray-based approach by specifying 3 types of rays [52]. The D-ray models the LOS and ground reflection in a map-based manner. Then in a stochastic approach, the R-ray represents the reflections from undetermined surroundings and the F-ray represents the reflections from moving objects like vehicles.

From the implementations and the trend, the modern channel models are bringing more consideration of the explicit environment, as a support for the 5G system, due to its increased sensitivity to the on-site scenario. However, these map-based models, are still based on simplified scenario structures, as a trade-off to the modelling complexity and computing efficiency.

2.2 Deterministic Radio Propagation Models

The deterministic models simulate the actual EM wave behaviour as propagates through the detailed modelled scenario, unlike the empirical models which are based on

stochastic assumption and measurement tuning for categorized scenarios. The deterministic models can predict both time and spatial propagation parameters more accurately than empirical models. For the relatively high accuracy of simulations, the deterministic propagation models are currently widely used in radio propagation modelling software. However, the high complexity of the deterministic models leads to increased time consumption and hardware requirements, such as memory and clock frequency. Thus, to address the bottleneck of speed, some studies have put efforts on simplifying the modelling methods. In this section, the differential equation solvers, ray-tracing and ray-launching based studies will be discussed.

2.2.1 Differential Equation Solvers

FDTD was initially developed by Yee in 1966 [53] based on Maxwell's equations. By solving the equations in the time domain using finite-difference approximations, the method is versatile in different scenarios. Also, FDTD is accurate [54] and accounts for the effect of refraction, reflection and diffraction. From then on, the method has been widely used to simulate the EM propagations in small scopes.

With the era of mobile communication and the need for network planning, attempts had been made to apply the FDTD in indoor and urban scenarios. However, due to its large memory consumption and calculation time, simplification has to be made, for example reducing the dimension to 2D [55]. A different approach is to combine the FDTD with a ray-tracing method, in which the ray-tracing is used to provide long-range calculation while FDTD is used to deal with the complex environment in the close range [56].

The complexity of FDTD is bound to its grid resolution, which is capped by the modelled wavelength. Every voxel in the grid needs to be allocated in memory and

processed independently. Thus, the usage of FDTD in large scenarios especially urban outdoor is rather limited [57]. Studies have to lower the modelled frequency for a longer wavelength to make the calculation feasible [58]. Another impact to the speed is due to the calculation in the time domain, that thousands of iterations could be needed to propagate through the voxels.

Recently, the introduction of hardware acceleration has drawn the attention of its application on FDTD. An implementation simulated in an urban environment has shown a speed increase of around 100 times [59]. However, the calculation is still based on 2D, and a frequency of 15MHz is used to cope with memory consumption. Not to mention that the availability of GPU memory is more critical than CPU memory. Another implementation in [58] presented a similar improvement of around 100 times with femtocell simulation, but a lower frequency is still used.

Another time-domain numerical solution is introduced in [60] based on TLM. From the similarity between wave propagation and transmission line theory, the TLM methods approximate the range of interest into a mesh of transmission lines to simulate the EM fields in different dimensions. Its difference with FDTD is more about the modelling approaches, i.e. a physical transmission line model versus a mathematical model based on differencing [61].

While part of the limitation is due to the many iterations needed in the time domain, a frequency domain method is proposed in [62] by transposing the parflow algorithm, which is similar to TLM, into the frequency domain parflow method (FDPF). Its implementation has shown the potential in indoor radio estimation, as the calculation within the range of a building is finished in 45s [12]. Although a 30min pre-processing

stage is needed before the calculation, it is less an issue for network planning since the pre-processing cache can be reused for different antenna placements in the same scenario.

2.2.2 Ray-tracing based Deterministic Model

The ray-tracing approach uses the ray model to represent the radio propagation in space. This is an effective approximation for high frequencies especially for 5G since the behaviour of EM wave is approximating the light. The transmitted EM waves are represented by rays from the transmitter to receivers with the following assumptions [63]:

- The trajectories of the rays are straight lines in a homogeneous medium.
- The rays follow the laws of the propagation mechanisms, including LOS, reflection, penetration and diffraction.
- The rays carry energy, which is modelled by radiation through infinitesimally small tubes.

With the concept of the rays, the EM waves can then be classified by their propagation mechanisms including LOS, reflection, refraction and diffraction. For the ray propagating in a complex scenario, the mechanisms mentioned above will be modelled by GO when the ray meets obstacles in the environment.

The reflection and refraction happen when a surface is hit. The Law of Reflection and Snell's Law are used to determine the direction of the newly generated reflection ray and refraction ray. In addition, the EM field strength needs to be computed with the incident polarization and corresponding reflection and transmission coefficients, which are provided by the Fresnel formulas [64, 65]. When a ray propagates from air to the medium with a higher refractive index than air, the E-field transmission coefficient and reflection

coefficient for transverse electric (TE) and transverse magnetic (TM) polarisations are given by

$$\Gamma_{\mathrm{TE}} = \frac{\cos\theta - \sqrt{\varepsilon_r - \sin^2\theta}}{\cos\theta + \sqrt{\varepsilon_r - \sin^2\theta}} \tag{2-2}$$

$$\Gamma_{\mathrm{TM}} = \frac{\varepsilon_r \cos\theta - \sqrt{\varepsilon_r - \sin^2\theta}}{\varepsilon_r \cos\theta + \sqrt{\varepsilon_r - \sin^2\theta}} \tag{2-3}$$

$$T_{\mathrm{TE}} = \frac{2\cos\theta}{\cos\theta + \sqrt{\varepsilon_r - \sin^2\theta}} \tag{2-4}$$

$$T_{\mathrm{TM}} = \frac{2\sqrt{\varepsilon_r}\cos\theta}{\varepsilon_r \cos\theta + \sqrt{\varepsilon_r - \sin^2\theta}} \tag{2-5}$$

where θ denotes the incident angle.

If a ray meets around the edge of an obstacle, the diffraction will be triggered. The uniform theory of diffraction (UTD) [66] is commonly used on the computation for the diffraction scenarios [67], from which the diffracted rays are forming a cone circling the edge with the diffracted angle equals to the incident angle, as shown in Figure 2-3. The diffraction is expensive to model, as its complexity is similar to modelling another transmitter in the scenario. Therefore, some ray-tracing models omit the modelling of diffraction behaviour [58], and some other models use empirical approaches to account for diffraction [21].

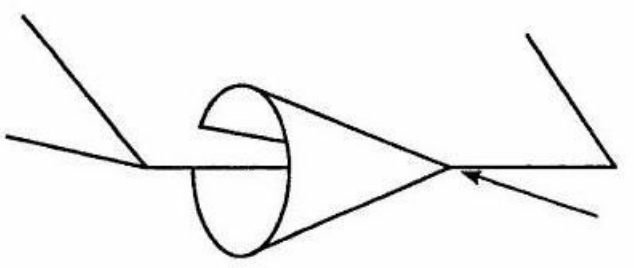


Figure 2-3 Diffracted paths from an incident ray hitting the edge

After all, the accuracy of the ray-tracing method requires the thorough geometrical and EM description of the scenario [68], which means that the more complicated the target environment is, the more computing time and computer memory are required. In the literature, this problem is addressed from both perspectives of accuracy and computational speed.

For indoor scenarios, when the transmitter and the receiver are placed close to complex lossy structures with finite dimensions, such as generic structural components and furniture, the scattered fields are hard to be obtained by ray-tracing. As mentioned in the previous section, a practical solution to improve the accuracy of the ray-tracing techniques is the combination with the FDTD method. When it comes to the area close to the complex discontinuities, FDTD is employed while the most area in the region is computed by ray-tracing. The derivations under the hybrid method can significantly improve the accuracy of prediction while maintaining an acceptable occupation of the computational resources [56].

To improve the computational speed of ray-tracing, the algorithms for determining the trajectory of the rays from the transmitter to receiver efficiently have been developed in the literature. Fundamentally, Fermat's principle of least time [5] is used to decide the path for a ray from the transmitter to the receiver. The imaging method [69, 70] is

commonly used in ray-tracing to find out the trajectory of the rays. It generates the images of the ray at all planes and then considers these images as the secondary sources for the subsequent points of reflection. A more efficient method has been discussed in [71], named the visibility tree. It establishes the visible tree recursively following the basic geometrical principles within a predefined maximum order of propagation paths considered in the ray-tracing model. With this algorithm, all the objects on each level in the tree are directly visible from the transmitter from the previous level. Therefore, all the propagation paths from the original transmitter to each reception point through reflection and diffraction, are fully defined before computations. Then, the computational cost could be reduced with the prepared visible tree [72]. Furthermore, the extension has been made in [73, 74] to accommodate the application in the 3D environment. A ray tube tree is constructed recursively through interactions to obtain the ray paths in 3D space. With the conversion from 2D to 3D, ground reflections and the finite heights of obstacles, like buildings in the urban environment, are appropriately addressed.

The ray-tracing algorithm provides relatively high prediction accuracy on EM wave propagations. It has been widely employed in different fields such as indoor localization [64] and network deployment optimization with the combination of optimization algorithms such as particle swarm optimization (PSO) in [75] and genetic algorithm (GA) in [76]. Although, the ray-tracing based algorithms are applicable for simulating all CIRs in the entire 3D scenario, usually only a selection of locations are computed. Its performance could be a critical issue if the number of selected locations is large. Not to mention the studied problem which is to provide the full 3D volume calculation.

2.2.3 Ray-launching based Deterministic Model

The ray-launching method also referred to as brute-force ray-tracing, shoots and bounces a large number of rays from the transmitter exhaustively for all the possible reception locations [77]. While its basic assumptions and propagation principles are the same as ray-tracing algorithms, the major difference between ray-launching and ray-tracing is that if rays are traced specifically to predetermined positions of the receiver. In ray-launching, the given environment will be discretized into unit sections representing the possible reception locations, and for a traced ray, its computed field will be updated and stored on every location it goes through [78]. Like the ray-tracing method, the works related to ray-launching are also focusing on the improvement of accuracy or speed.

In order to improve accuracy, the extension or compensation models are developed in the literature. The measured data in urban scenarios is used to calibrate the predictions in [79]; the influence by human bodies is considered to facilitate a differential description of the time-variant scenario in the indoor environment in [80]; A directive diffuse scattering model is included in the ray-launching models and compared with the simulations and the measurement in a 5.4 GHz indoor environment in [81]; A hybrid method combining ray-tracing and ray-launching is used in [82], where the image method is employed to adjust the ray trajectory for the known transmitter and receiver positions with respect to a certain reflection surface.

As an important research area, different types of acceleration strategies for the ray-launching method are developed to achieve better computation efficiency. As the initial step for the ray-launching, the discretization of the environment impacts the computation significantly. Conventionally, the environment is divided uniformly into a grid with constant voxel size. The transmitters, receivers, and obstacles in the scenario are sorted

into specific voxels, then the rays propagate through the grid. A fast algorithm is proposed in [83] to decide the interceptions of rays and the grid. With the algorithm, the interceptions can be simply decided by two operations, an addition and a comparison. However, transmitters, receivers and obstacles are practically non-uniformly distributed. The computational speed could be degraded when employing the uniform division algorithm, therefore, the bounding volume hierarchy (BVH) method, which divides the environment into units with different sizes, is used for scenario discretization [84]. The selection of partitioning lines of the environment is significant, therefore the algorithms such as surface area heuristic (SAH) are developed to address this problem in [85]. Moreover, the triangular grid which divides the buildings into a mesh of triangles using 2D projection is introduced in [86]. By dividing the regions of interest into the triangular grid, it is easier to identify the trajectory of the rays since the number of possible choices for a ray to enter the next triangle is only 2.

Another perspective to speed up the computation is dimensionality reduction from 3D to 2D for a propagation environment. For example, the triangular grid in the 2D ground plane can be used to recover the 3D ray paths with a known height of the transmitter and receiver. Furthermore, the propagation environment can also be described in 2.5D format, which means that the obstacles in the environment are described by their projections on the ground and their height. Such simplification of the model helps the reduction of the computational burden. It shows relatively high accuracy when applying in the urban scenarios since the 2.5D format of the environment can still include comprehensive information. In [87], a cube-oriented ray launching algorithm (CORLA) is applied in field strength prediction based on a 2.5D environment model which offers the precision of the

ray-launching algorithm with a comparable time consumption to empirical models. This format of the simplification is also extended into the indoor scenarios in [64, 78] and compared with the prediction results obtained by the 3D model and empirical model. The prediction results are still acceptable, and the computation speed is improved by a large scale compared with the 3D model. The other type of algorithms for dimensionality reduction proposed in [22, 88, 89], also show good accuracy in predictions with improved efficiency. The maintained accuracy from dimensionality reduction is mainly due to that only the result on a specific height, e.g. the ground level or floor level, are concerned on the applications so far. Such an assumption could be less applicable for mmWave which is sensitive to the environment, and O2I scenario which highly depends on the variation on the z level. Also, irregular scenarios like stadium can hardly be expressed in 2D/2.5D format.

On the other hand, the computational complexity can be reduced by simplifying the ray modelling mechanisms. Some methods introduce a hard limit to drop a ray if a maximum number of interactions is reached, or if the signal strength on the ray is becoming lower than a threshold [1, 78, 90]. Depending on the actual setting, the simplification could be valid since the uncertainty of a ray increases with the number of interactions. Also, for very weak rays, their represented signal strengths may be below the sensitivity of a receiver, thus they can be dropped. Specifically, for the diffraction, due to the high complexity, it is also approximated in the literature. Simplified expressions of the four characterization factors of the diffraction coefficient are obtained by polynomial curve-fitting procedure through the least square criterion in [91]. Moreover, for flat-roofed buildings, the diffraction coefficients are expressed by an array for the prediction of the

multiple diffractions generated from finitely conducting buildings in [92]. The mathematical approximations of the UTD coefficients are also explored in [93]. The practical behaviours for different diffraction formulations are concluded in [94]. With the specific propagation scenarios and the approximations made on mathematical formulations, the propagation mechanisms could be simplified with a tolerable sacrifice of the accuracy, which facilitates the engineering usage of the theoretical models.

With the development of HetNet, the ray-launching techniques are also applied to characterize wireless system performance for I2O and O2I scenarios. Compared with the pure indoor or outdoor environment, the I2O and O2I scenarios are more complicated since not only characteristics of both indoor and outdoor scenarios should be considered but the penetration loss caused by the external structures of the buildings is also important. Therefore, the computation burden increases significantly due to the variation of the propagation environment details and expected result resolutions. In the previous works, the simulation including both indoor and outdoor scenarios is normally modelled by a hybrid method where different grid sizes or model methodologies are used. In [95], indoor rays are modelled by the 2D FDTD while the outdoor rays are modelled by the 3D ray launching algorithm. In [96], a full 3D joint indoor to outdoor ray launching algorithm with different resolutions for indoor and outdoor environments is proposed. The combination of the ray launching algorithm with other compensation methods to address the penetration loss caused by the building structures are also commonly seen in the literature, such as the physical optics (PO) model in [97].

However, these O2I approaches are in general trying to establish new radiating sources on the boundary between outdoor and indoor scenarios, due to the fact of using

hybrid methods. As a result, the transition between scenarios becomes discontinued. Thus, it could be difficult to represent the actual shadowing condition in the environment, which may be critical for mmWave. Therefore, a seamless approach during the transition would be more suitable.

Meanwhile, hardware acceleration is also starting to be applied in ray-based propagation methods. Significant improvement of the computation efficiency is achieved in [98, 99], by using GPU acceleration for 2D beam tracing. The acceleration method is also extended into 3D in [100-103], but they all limited the results on a few fixed reception spheres, therefore had no consideration of angular dispersion. This could make the implementation less complex but still sufficient for the validation against measurements, which is also a collection of fixed points. These methods are not suitable for the 3D volume calculation in this study. A tube-based method in which the tube model has a volume is proposed in [9]. Thus, the tube is continuous which doesn't bear the angular dispersion naturally. However, its GPU implementation takes more than 100s to calculate a single office with a dimension of $10m \times 12m$. Such speed doesn't match the hardware acceleration performance [104]. It could be the impact of additional complexity from modelling the tube. A model with a closer match to the advertised performance is developed in [105], in which the speed is 30MRays/s, but it is raster-based. After all, the application of hardware acceleration in a ray-launching model is still under desired, providing the full 3D volume prediction and the solution to angular dispersion.

2.3 Summary

In this chapter, the principles and studies for the empirical and deterministic propagation models are discussed. A comparison between them in terms of complexity, accuracy and required geometrical detail is given in Figure 2-4. In general, the empirical models are fast in calculation and easy in implementation, while acceptable accuracy is provided. But they are measurement and scenario dependent. The tuning process of an empirical model for many specific scenarios could be tedious. On the other hand, the deterministic models are versatile and more accurate. But they suffer from more computational burden and require detailed geometrical information.

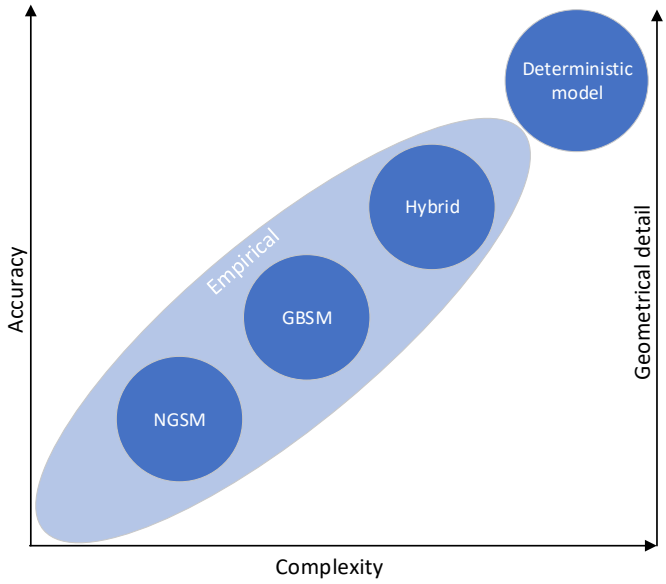


Figure 2-4 Comparison between empirical and deterministic models

However, rather than competitors, these two approaches could serve as compensators to each other. Like the introduction of empirical hybrid models, there is a trend of moving towards more deterministic simulations in the empirical approach. Also, some deterministic models are offloading part of the calculation to empirical methods, for

example, the knife-edge. Actually, for some the wave behaviours like scattering and reflection on dynamic objects, it is impractical to model them in a fully deterministic manner, due to the complexity. While these phenomena can be simulated much easier using empirical methods. Therefore, a future trend could be hybrid models dividing the consideration of large-scale components and small-scale components into deterministic and empirical approaches.

Yet for the large-scale components, the current deterministic models still lack an efficient solution for the problem studied in this thesis, which is to provide full 3D volume radio prediction in medium-range scenarios. Some candidates do consider the angular dispersion with fast calculation, but they are 2D/2.5D based or rasterized [11, 12]. Some other candidates provided promising speed using hardware acceleration, but the problem of angular dispersion is ignored. This study aims to develop an efficient solution towards the problem by a deterministic approach.

3 Compare Ray-tracing, Ray-launching and MR-FDPF for Medium-Range 3D Volume Radio Prediction

This chapter aims to find the preferred deterministic approach for the studied problem, which is to provide the full 3D volume radio prediction within medium range. The candidates are the three main streams, i.e. ray-tracing, ray-launching and differential equation solvers, for which the MR-FDPF will be used due to its presented speed that the 2D coverage calculation for building with size $100m \times 25m$ can be finished in 8.1 seconds[12]. Firstly, the complexity of the ray-tracing method will be evaluated in the 3D volume problem domain and compared against the standard ray-launching method. Then the identified high-performance ray-launching model, which is the intelligent ray launching algorithm (IRLA) [106], will be compared against the MR-FDPF model, by integrating both into a general building modelling tool iBuildNet [107]. Therefore, a side-by-side comparison will be available for the computing speed, memory consumption and result accuracy. In the end, the preferred method will be drawn and used for the subsequent study.

3.1 Speed Comparison between Ray-tracing and Ray-launching by Algorithm Complexity

Ray-tracing and Ray-launching are both ray-based modelling methods, the fundamental difference between them is the ray casting strategy, i.e. the ray-tracing traces ray back from the PoI to the transmitter, while the ray-launching does a brute force launching of massive rays from the transmitter. Ray-tracing is a point-to-point method,

which indicates that it is not suitable for solving the 3D volume problem [106]. Also, its computation complexity is exponential to the number of interactions, while it is linear for the ray-launching [7, 108]. However, this conclusion didn't consider the complexity of solving the angular dispersion in ray-launching, which should be significant. Since the ray-tracing method does not have this problem, it is fair to repeat the comparison with ray-launching addressing the angular dispersion.

3.1.1 Complexity in 3D Free Space

By sampling the interesting space into a grid with uniform voxels, the 3D volume calculation can be represented as providing the pathloss of each voxel in the grid. As shown in Figure 3-1, the voxel is uniform if $i = j = k$, which can be regarded as the ray collection resolution. Also, given both methods are ray-based, a relative algorithm complexity can be represented by the total number of rays needed to be cast in each method. Therefore, their complexity in a free space environment can be deduced.

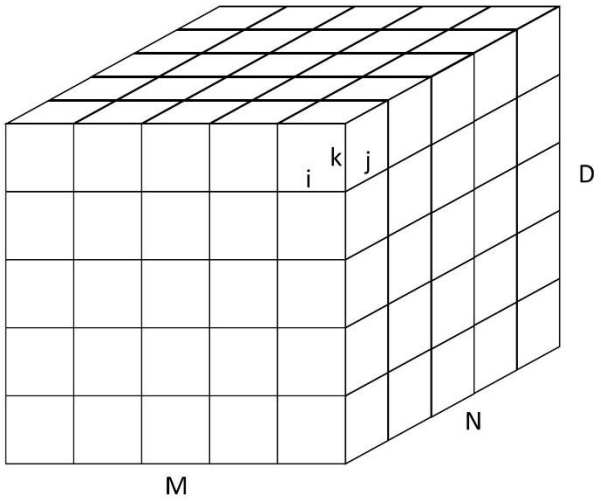


Figure 3-1 An example of a 3D grid.

For ray-tracing, each voxel can be regarded as a PoI and needs to be traced back to the transmitter. Thus, the total number of needed rays is equal to the total number of voxels which is

$$Num_T = M \times N \times D \tag{3-1}$$

where M, N and D are the number of voxels on x, y and z dimensions.

For ray-launching, the full grid can be covered as long as a ray is cast to each voxel on the boundary [87]. Therefore, the total number of rays needed is equal to the number of voxels on the grid surface, which is

$$Num_L = M \times N \times 2 + 2 \times (M + N - 2) \times (D - 2) \tag{3-2}$$

Assume the grid is a square cube where the $M = N = D = l$, then the complexities of these two methods become

$$Num_T = l^3 \in O(l^3)\ as\ l \to \infty \tag{3-3}$$

$$Num_L = 6l^2 - 12l + 8 \in O(l^2)\ as\ l \to \infty \tag{3-4}$$

Therefore, the complexity of ray-tracing is one power higher than the ray-launching method in free space.

3.1.2 Complexity in Space with Single Wall

The complexity increases with interactions, i.e. the number of walls in the space. Starting with a single wall for ray-tracing, the setup effectively creates a mirrored transmitter against the wall. Therefore, in addition to the free space case, the same number of rays have to be cast to the mirrored transmitter. To simplify the comparison of complexity, the following scenario will always assume a square cube grid. In this case, the total number of rays become

$$Num_T = 2l^3 \in O(l^3)\ as\ l \to \infty \tag{3-5}$$

For the case of ray-launching, instead of mirroring the transmitter, the wall will mirror the space. Depending on the location of the wall, the mirrored space could have an overlap with the original space. In the end, the original space combined with the mirrored space will form the total space which denotes the problem complexity.

The size of the total space depends on the location of the wall. In case the wall is right in the middle of the original space, then the mirrored space will fully overlap the original space, thus the total space equals the original space. In case the wall is on the boundary surface of original space, then the mirrored space will have no overlap with the original space, thus the size of total space will be double the size of original space. Assume the wall is located on the xoz plane with a distance d to the centre of the space, where $0 \leq d \leq l/2$, then the size of total space can be represented as

$$Size_{Tot}(d) = \left(1 + \frac{d}{l/2}\right) \times Size_O, \tag{3-6}$$

where $Size_O$ is the size of the original space.

Therefore, the task of ray-launching becomes casting a ray to each voxel on the boundary of the total space. Then the total number of needed rays can be approximated by

$$Num_L(d) \in O\left((6l^2 - 12l + 8) \times \left(1 + \frac{d}{l/2}\right)\right) \in O(l^2)\ as\ l \to \infty \tag{3-7}$$

It can be seen that the wall will increase the total number of rays for both methods, but the magnitude of complexity doesn't change. This is because only one wall is assumed,

which has a limited impact. The complexity dependency on the number of walls will be discussed next.

3.1.3 Complexity in Space with Dense Walls

To consider the impact of walls, the worst case is the dense urban scenario, in which the building blocks have a high density. To help the analysis of the complexity in such a scenario, it is reasonable to assume uniformly distributed walls within the scenario.

3.1.3.1 Ray-tracing

For the ray-tracing, given K walls in total, each of the walls will create a mirrored transmitter. Thus, there will be K mirrored level 1 transmitters, where level 1 denotes that they are mirrored once. Then each level 1 transmitter can be mirrored again by the rest $K-1$ walls, and this will give $K(K-1)$ level 2 transmitters. Many methods limit the maximum number of interactions [1], which effectively is the number of levels in this case. So, for level N, i.e. the Nth interaction, the number of mirrored transmitters will be

$$Num_{tx}(N,K) = \prod_{n=0}^{N}(K-n) \tag{3-8}$$

Then the total number of transmitters for N interactions will be

$$Num_{total\_tx}(N,K) = 1 + \sum_{m=1}^{N} Num_{tx}(m,K) \tag{3-9}$$

In the end, by combining equation (3-5) with equation (3-9), The total number of rays needed for ray-tracing is

$$Num_T(N,K) = l^3 \times Num_{total\_tx}(N,K) \tag{3-10}$$

From equation (3-8), it can be observed that the number of rays is dominated by the number of transmitters in the Nth interaction. So, the complexity of the method can be deduced by

$$Num_T(N,K) \in O\left(l^3 \times \prod_{n=0}^{N}(K-n)\right) \in O(l^3 K^N) \tag{3-11}$$

Equation (3-11) proves that the complexity of ray-tracing has an exponential dependency on the number of interactions with base *K*, which is the number of walls. The conclusion meets the result in [7]. In addition, the equation also shows that the complexity has a dependency on the number of voxels in the grid.

It's worth noting that in practice, for the level 1 mirror, the *K-1* walls won't all be visible through the mirror, as the wall size is not infinite. However, there is no prior knowledge about the visibility condition between walls. Some methods proposed a pre-processing stage to reveal the visibility [72], so as to reduce the number of walls that need to be mirrored. Still, this will, on the one hand, consume computing power during the pre-processing, on the other hand only reduce the base from *K* to *K-x* in equation (3-11), which does change the magnitude of the complexity.

3.1.3.2 Ray-launching

For ray-launching, the number of rays is decided by the surface area of the total expanded space as discussed in Section 3.1.2. When there are many walls in the space, the worst case is that the space will be fully mirrored in each direction. In this case, for one mirror, the total space becomes a square cube with a length of $3l$. Then each mirrored wall in the $1st$ mirrored space can do the mirror again, but only the entities in the same mirrored space can be mirrored. The worst-case will expand the total space by l again on each direction, and results in a square cube with a length of $5l$. It's worth noting that, the number

of mirror actions is effectively the number of interactions. So, for the Nth interaction and the worst case, the size of total space will be

$$Size_{Tot}(N) = (2N+1)^3 l^3 \tag{3-12}$$

Therefore, by using equation (3-4), the number of rays needed for ray-launching is decided by the number of voxels on the boundary, which is

$$Num_L(N) = 6(2N+1)^2 l^2 - 12(2N+1)l + 8 \tag{3-13}$$

And the complexity of ray-launching could be approximated as

$$Num_L(N) \in O(N^2 l^2) \tag{3-14}$$

By comparing equation (3-14) with equation (3-11), it is clear that ray-launching does have much less complexity than the ray-tracing for solving the 3D volume problem. In the meantime, there are two findings worth discussion.

Firstly, for ray-launching, the complexity has a dependency on the number of interactions N by N^2, rather than linear as given in [7]. As mentioned earlier, this is due to that the angular dispersion problem wasn't considered when conducting the conclusion of linear. This is also reflecting that solving the angular dispersion will introduce a significant among of complexity.

Secondly, equation (3-14) is indicating that the complexity of ray-launching has no dependency on the number of walls, which is against the intuition at first thought. It is due to that the ray-wall intersection problem isn't considered in the deduction above. Actually, the ray cast in ray-launching method needs to have an intersection test against each wall in the scenario. While for ray-tracing, such problem is less complex as each mirrored transmitter knows the wall it is mirrored from. However, many space indexing methods simplify the intersection problem significantly, like uniformed grid and BVH [84]. The

additional complexity introduced by the intersection test will be evaluated in the next chapter with a chosen space indexing method.

After all, although the complexity of ray-launching in higher than anticipated in literature [7], due to the angular dispersion problem. It is still much less complex than the ray-tracing method. Therefore, the ray-launching should be the preferred method over the ray-tracing for providing the 3D volume prediction in this study.

3.2 A Side-by-side Comparison between the MR-FDPF Model and the IRLA Model

The MR-FDPF model is a differential equation solver in the frequency domain, and the IRLA model is a ray-launching model in 3D rasterized environment. A brief introduction of both models is given in Section 2.2. While both have presented fast 3D volume radio prediction for the small cell within minutes, the purpose of this section is to understand their advantages and disadvantages from each other, and their fitness to the medium-range 3D volume calculation. Therefore, they are integrated into a 3D building modelling tool to give a side-by-side comparison for the accuracy and speed.

3.2.1 Algorithm for MR-FDPF Model

The original TLM method approximates the scenario into a mesh grid, in which the link between two neighbour nodes can be modelled as a transmission line circuit. Thus, for a 2D model, each node will be linked to its neighbours by 4 circuits, and for a 3D model, it will be 6 circuits. Then the EM field could be transmitted, reflected and absorbed by the circuits according to the transmission line theory [109]. Thereafter, the scenario and the physical structure can be approximated into the circuits with corresponding TL

components. In the end, the propagation of the EM field in the transmission line network can be simulated in iterations over time.

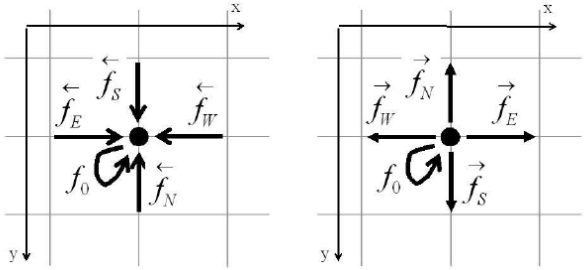


Figure 3-2 The inward and outward flow on a node in the Parflow method

Similar to the TLM, the Parflow method models the electric field as flows propagating between nodes in a grid. A 2D representation of the flows on a node is shown in Figure 3-2, where the field is divided into 4 flows on the directions W, E, S, N, and f_0 is a stationary flow representing different mediums. Then the outward and inward flows can be given as $\overrightarrow{F} = \left(\overrightarrow{f_E}, \overrightarrow{f_W}, \overrightarrow{f_S}, \overrightarrow{f_N}, f_0\right)^T$ and $\overleftarrow{F} = \left(\overleftarrow{f_E}, \overleftarrow{f_W}, \overleftarrow{f_S}, \overleftarrow{f_N}, f_0\right)^T$. Thus, the time domain field equation at location r can be written as

$$\overrightarrow{F}(r,t) = \Sigma(r) \cdot \overleftarrow{F}(r, t-dt) + \overrightarrow{S}(r,t) \tag{3-15}$$

where $\overrightarrow{S}(r,t)$ denotes the inward contribution from a source, and $\Sigma(r)$ is the local scattering matrix. Then by applying a Fourier Transform, the corresponding frequency-domain equation can be given as

$$\overrightarrow{F_e}(r) = \Sigma_e(r) \cdot \overleftarrow{F_e}(r) + \overrightarrow{S_e}(r) \tag{3-16}$$

where $\Sigma_e(r) = \Sigma(\mathrm{r}) \cdot e^{-j2\pi\omega dt}$. By solving the equation using series expansion, the MR-FDPF model computes the electric field on each point at a given resolution. The majority of the complexity is for the computing of the scattering matrix.

3.2.2 Algorithm for IRLA Model

The IRLA as a ray-launching model also includes a few additional mechanisms including rasterization, 3D to 2.5D projection and gap filling. These mechanisms are introduced to reduce the algorithm complexity and solve the angular dispersion.

During the rasterization stage, the characteristics of the entities within the environment are sampled into voxels in a uniformed grid. Each voxel is assumed to be homogeneous and would only represent one entity, e.g. one wall. Therefore, the problem of ray polygon intersection test can be saved by doing the ray grid traversal. And the sampling of the ray into pathloss result can be performed at the same time. According to [90], around 90% of the computing time is spent on the intersection test, thus the save from rasterization would be significant. However, such simplification would be at the cost of losing accuracy, as the original vector data are sampled, and some entities may need to be discarded if multiple entities appear in the same voxel.

Then during the ray propagation simulation, the 2.5D projection will cast a ray in 3D coordinates into the horizontal 2D plane after the first interaction [11]. Thereafter, all the interactions including diffraction will happen only on the 2D plane. Such projection will again reduce the complexity significantly by one dimension, but at the cost of accuracy, since all the subsequent interactions that could happen in 3D are neglected. Such simplification could be less an issue in an indoor scenario as a floor is bounded by the ground and ceiling, thus that propagation of rays is more likely on the horizontal level, especially when it is far from the transmitter.

The IRLA does provide a solution for the angular dispersion, by interpolating the gap between separated rays [110]. After a reflection, if the gap between two neighbour rays becomes larger than a resolution, then interpolated rays will be added in between them to

fill the gap. Thus, all the gaps will be filled by the interpolation, and since the operation is performed in 2D, the corresponding complexity will be much less than a 3D solution.

In the end, the IRLA has presented the result with no angular dispersion by interpolation. Meanwhile, the simplification provided by rasterization and 2.5D projection will reduce the ray-launching complexity significantly, so its calculation of a small cell can be finished within a minute. Therefore, the speed of IRLA is shown to be suitable for providing 3D volume radio prediction.

3.2.3 Measurement Campaign

The used scenario is one floor in an indoor building with a dimension of $49.9m \times 34.2m \times 3m$, and the floor plan is given in Figure 3-3. The external walls of the building are made of concrete and the internal rooms are separated by walls made from bricks. In addition, the doors and windows are made of wood and glass respectively. Both models supply default materials for the above types, but it is hard to determine the exact property values. Thus, calibration by the measurement is needed.

The measurement data are a collection of received signal levels on the locations marked by the red dots in Figure 3-3. The R&S®FSH handheld spectrum analyser [111] holding around 1.5m height is used to measure the data. An EIRP of 20dBm is used and the frequency is set to 3.4GHz to avoid interference with Wi-Fi signal in the office environment. The measured bandwidth is 300kHz and the sensitivity of the analyser is -141dBm/Hz. Therefore, the sensitivity with the bandwidth can be given as

$$\left(-141 + 10log(3 \times 10^5)\right) \approx -86dBm \tag{3-17}$$

Thus, the measurement points measured with signals below -86dBm are discarded to separate noise.

Figure 3-3 Floor plan of the indoor office with measurement locations

3.2.4 Calibration Mechanism and Result

The material database used by a propagation model could be calibrated against measurements, as the material property may not match the real scenario. The IRLA has supplied the calibration function using the Simulated Annealing (SA) algorithm. The SA can avoid local optimization for nonlinear problems. However, the MR-FDPF does not come with calibration originally, thus GA [112] is implemented to provide the function.

The GA is a generalized optimization method which models the procedure of genetic evolvement through generations. For each generation, a set of solutions will be formed, which is represented by chromosomes. Specifically, for the MR-FDPF, the refraction index and attenuation coefficient of each material are combined as a chromosome. Thereafter, the chromosomes are evaluated by a fitness function, which calculates the root mean square error (RMSE) between the prediction result and the measurement. Then the chromosomes with better fitness values are allowed to produce the next generation. The evolvement will be stopped once a predefined max number of

iterations is achieved. As the time consumption of MR-FDPF is majorly on the pre-processing of the scenario, the processed data can be cached and reused for the fitness function evaluation. Thus, the calculation of many generations could be feasible in speed.

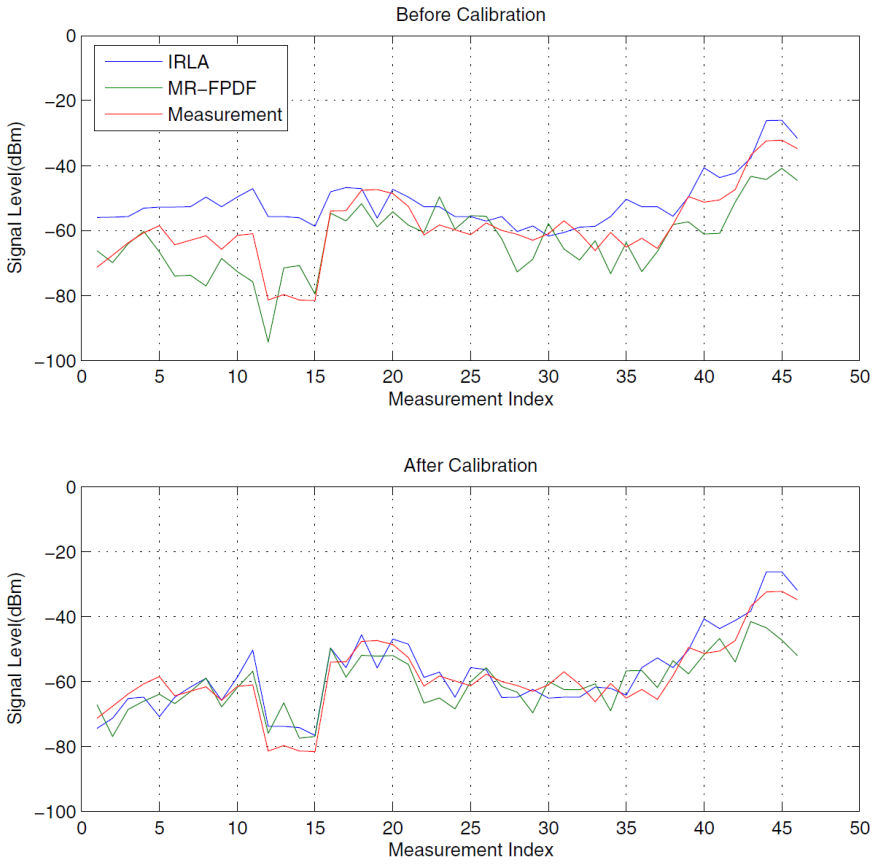


Figure 3-4 Calculation results comparison before and after calibration

The calculated signal levels before and after calibration for both models are shown in Figure 3-4, with respect to the measured values. With calibration, the RMSE of IRLA is reduced from 10.46dB to 5.38dB, and for MR-FDPF it is reduced from 7.83dB to 6.3dB.

3.2.5 Speed Performance and Result Accuracy Comparison

The calculation is repeated for both models at different resolutions, which are 0.1m, 0.2m, 0.3m and 0.5m. And the results as received signal strength using 0.1m resolution is

shown in Figure 3-5, where the corresponding measurement points are shown as the coloured dots as well. Given the same scenario, the resolution will determine the number of voxels which will directly affect the complexity especially when both models are raster-based. Using a PC with Intel i7-3610QM CPU and 8GB RAM, the time and memory consumptions at different resolution are listed in Table 3-1. From the comparison, it is shown that the speed of MR-FDPF is faster than IRLA at all resolutions, especially at the high resolution 0.1m. However, the IRLA always consumes less memory than MR-FDPF, around a ratio of half. But it should be mentioned that the MR-FDPF is a 2D model while the IRLA is a 2.5D model.

	0.1m	0.2m	0.3m	0.5m
Time (IRLA)	9min44sec	43sec	8sec	3sec
Time (MR-FDPF)	38sec	8sec	4sec	2sec
Memory (IRLA)	1.02GB	205MB	94MB	42MB
Memory (MR-FDPF)	1.9GB	448MB	201MB	82MB

Table 3-1 Time and memory consumptions for both models at different resolutions

The RMSE values for both models are compared in Table 3-2. In overall, the accuracy increases with higher resolutions for both models. The IRLA is having less RMSE than MR-FDPF at all resolutions. This could be the improved accuracy from a 2.5D model over a 2D model. Specifically, when the resolution becomes worse than 0.3m, the drop of accuracy for MR-FDPF is rapid. This is due to the limitation of differential equation solver that the resolution is limited by the wavelength. For MR-FDPF, the resolution is suggested to be smaller than $\lambda/6$ [12], where λ is the wavelength.

	0.1m	0.2m	0.3m	0.5m
RMSE (IRLA)	5.1dB	5.4dB	5.8dB	6.9dB

RMSE (MR-FDPF)	6.5dB	6.8dB	8.6dB	11.5dB

Table 3-2 RMSE for both models at different resolutions

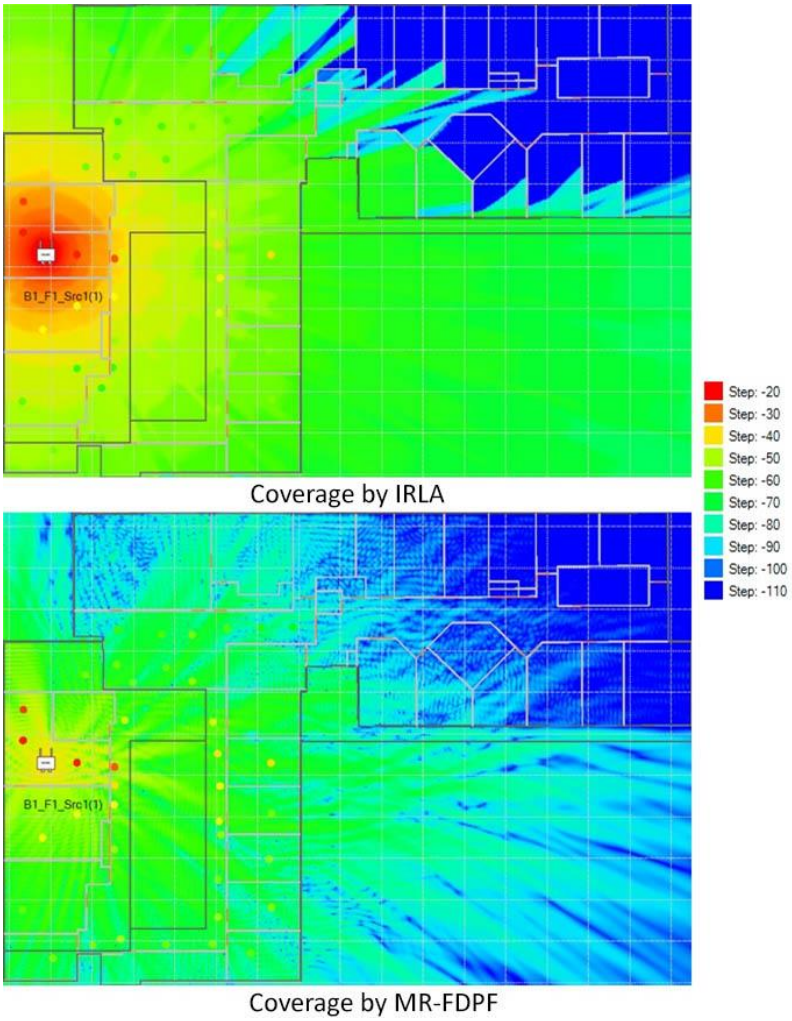


Figure 3-5 Calculation result by IRLA vs MR-FDPF at 0.1m

3.2.6 Discussion and conclusion on the comparison

In overall, both models are showing advantages in different aspects. The MR-FDPF is much faster than IRLA at high resolutions. This makes it more suitable for indoor fine calculations. However, its accuracy at 0.1m is still lower than IRLA at 0.3m. Taking their time consumption into consideration, the IRLA is actually faster if it is to provide the same accuracy regardless of used resolution. On the other hand, the accuracy of MR-FDPF

decreases vastly with the drop of the resolution, due to its restriction by the wavelength. This would limit its application in urban cell coverage prediction, in which such small resolution would be difficult to handle memory-wise and speed-wise. Not to mention the modelling of 5G mmWave would introduce more severe restriction. Therefore, the ray-launching approach is showing its advantage in 3D volume calculation, due to less dependency on the wavelength and less complexity when extending to 2.5D/3D.

Although the IRLA is showing good accuracy in indoor scenario, its simplifications may not be directly applicable in outdoor scenarios, especially the 3D to 2D projection. The model does adopt a hybrid approach using the knife-edge diffraction method for the large urban scenario [11], but as discussed in Section 2.1.1, this empirical approach is less accurate for small to medium ranges. Also, the impact of terrain elevation is simplified in the knife-edge method. With the application of mmWave in small cell, the corresponding coverage quality would be more sensitive to the exact geometrical conditions. Also, the usage of rasterized environment would have a limitation in resolving dense environment entities. For example, if two walls appeal in one resolution which could be 5m for outdoor, then one wall will be discarded.

Therefore, although IRLA has presented its speed capability and its accuracy in indoor, like many proposed methods, it puts the effort in simplifying the modelling method and detail. It is still desired to develop a fully GO-based deterministic model, that is vector based and capable of providing fast 3D coverage prediction.

4 A GO-Based Ray-launching Model (GORL) with Hardware Acceleration

In this chapter, a ray-launching model will be built with hardware acceleration to investigate the efficiency and solve the angular dispersion problem as discussed in previous chapters. Ideally, the principle of the ray-launching method is based on classic GO. However, even ray-launching turns out to be the preferred method among the 3 main streams as concluded in Chapter 3, the complexity is still hard to handle. As discussed in Section 2.2.3, lots of research put efforts on simplifying the ray-launching strategies. Compromise to accuracy has to be given as a drawback. Some methods also do not consider the problem of angular dispersion, resulting in not applicable for 3D volume prediction.

In the meantime, hardware acceleration is becoming mature and efficient in processing ray-based tasks. On the one hand, various gaming engines are fundamentally ray-tracing engines. On the other hand, the hardware manufacturers start to provide public APIs to help the usage of graphic card power in other computing domains. Currently, there are two well standardized platforms, CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language). Nvidia also provided the generalized ray-tracing engine, OptiX, which can process up to a few hundred million rays per second [104]. Therefore, this study will try to solve the challenges in a ray-launching model by using hardware acceleration, instead of making compromises, given the computing power indicated above.

This study is distinguished from existing literature by the selection of techniques and target, which is to provide fast 3D coverage prediction using vector-based approaches,

as concluded at the end of Chapter 3. More specifically, the prediction of a small cell coverage should be finished within minutes in an urban area. To the best knowledge of the author, the existing studies using the same techniques showed performance no close to the target. In the early days, the calculation of received rays at a single receiver location could take 15 minutes in a simplified scenario where buildings are represented by boxes [113]. A recent study using GO-based 3D ray-launching spent 36 hours to calculate an indoor office with an area of $91m^2$ [114]. Another simulation performed in outdoor with one base station spent 70 minutes and 720 minutes in two different scenarios [115]. The propagation models with performance closer to the target are only achieved by certain simplifications, for example, the IRLA introduced in Chapter 3.

Despite the speed of the existing ray-launching models, the performance target could be possible with the joining of hardware acceleration as discussed above. However, as concluded at the end of Section 2.2.3, most of the current adoptions of hardware acceleration in literature are limited to a few reception spheres, i.e. they are ray-tracing based. While for the ray-launching based ones, [9] and [116] are not showing expected performance boost from hardware acceleration, and [105] is raster-based. The knowledge barrier between high performance computing (HPC) and wireless communication could be the reason for some limited implementations and performances in the literature. There also seems a trade-off between the algorithm complexity and the delivered performance boost from hardware acceleration.

Meanwhile, another uniqueness of this study is the choice of OpenCL acceleration platform for a fair comparison. The existing publications using hardware acceleration for ray-based propagation models are all found to be based on the CUDA platform. The only

one found OpenCL adoption is based on empirical models [117]. However, as introduced in Section 4.1, CUDA is limited to be used on Nvidia GPU card. Thus the comparison for a pure performance improvement from hardware acceleration is not fair, as the original not accelerated program has to be run on the CPU which is a different device. Also, the device limitation of CUDA could be less preferred for industrial usage, since not every machine has an Nvidia GPU card, while OpenCL has much wider device support. The choice of CUDA in existing literature may be due to its ease of use when implementing the program.

Therefore, this study will start with a GO-based ray-launching model powered by the OpenCL hardware acceleration, firstly to understand the speed boost that can be delivered from hardware, then to exam the feasibility of solving the 3D radio prediction problem by the approach.

4.1 Introduction and Choice of the Hardware Acceleration Platforms: CUDA vs. OpenCL

Currently, the power of hardware acceleration can be provided by two platforms: CUDA and OpenCL. CUDA is a massive parallel computing platform provided by Nvidia [118]. Unlike the generalized central processing unit (CPU) which is good at processing many complex logic switches, the graphics processing unit (GPU) maximizes its performance at a large number of similar operations with fewer logic switches. This was originally designed to improve the graphic rendering efficiency, which is a problem of rendering many pixels in parallel. A brief comparison between the CPU and GPU architectures is shown in Figure 4-1. The CPU contains a large control block which is responsible for logic switches, together with a few algorithm logic units (ALU) which is

responsible for performing the actual calculation instructions. On the other hand, the GPU maximize its calculation power with massive ALUs with the trade-off of fewer control units.

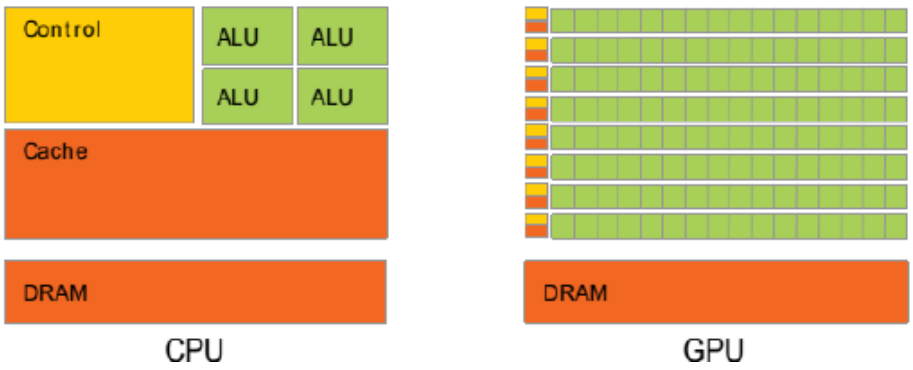


Figure 4-1 CPU vs GPU Architecture[119]

Therefore, the GPU architecture looks much more suitable for the ray-launchings algorithms, which is a problem of processing massive rays in parallel. However, if the model of ray becomes complex with quite a few logic switches involved, the suitability of such an algorithm may start to shift to the CPU side. This could be the reason why the hardware acceleration performance is not ideal when the ray model is complex, as discussed at the beginning of this chapter. A solution to such an issue is to simplify the ray calculation tasks given to the acceleration platform while keeping the complexity logic handled on the CPU side. A detailed solution will be described later in this chapter.

In the meantime, the OpenCL, maintained by the non-profit Khronos Group, is another parallel computing platform providing very similar APIs and features to the CUDA [120]. The major difference is the independency of OpenCL, i.e. CUDA program can only run on Nvidia devices, while OpenCL program can be executed in various types of devices including CPU, GPU, digital signal processor (DSP) and field-programmable gate arrays (FPGA) [121]. Figure 4-2 and Figure 4-3 show very similar architectures between the

CUDA and OpenCL. The major difference is the presence of texture memory in CUDA, which is used for storing read-only data in GPUs, while since the OpenCL is supporting various devices apart from GPUs, there is no texture memory modelled.

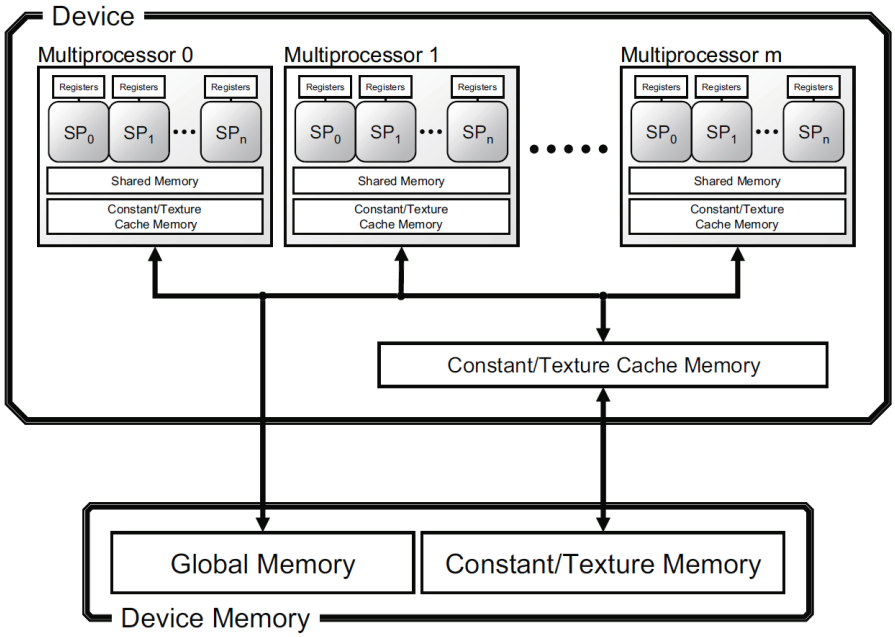


Figure 4-2 CUDA GPU Architecture [122]

Performance-wise, although many experiments are showing a faster speed from CUDA than from OpenCL generally with a ratio of 130%, the actual efficiencies of these two platforms are similar under fair comparisons [123]. On the other hand, a few tools are porting the programs between OpenCL and CUDA due to similar APIs and architectures [120, 124].

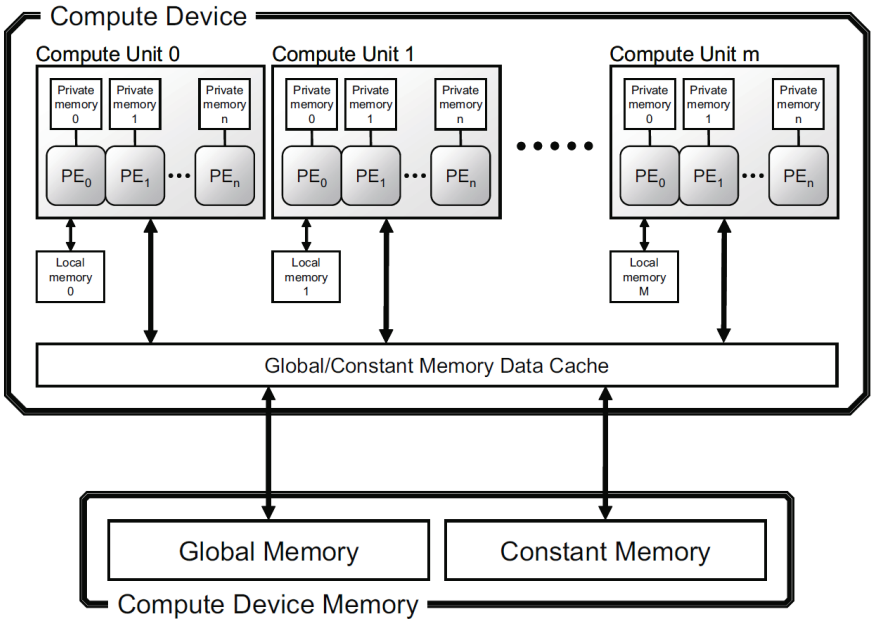


Figure 4-3 OpenCL GPU Architecture [122]

Therefore, this study chooses OpenCL due to its openness in device selection, and similar performance to the CUDA. This choice will give an advantage for comparing the CPU implementation and the OpenCL implementation on the same CPU device to reveal the fair improvement from hardware acceleration, which is not possible with CUDA as only GPU is supported. Also, the comparison with other existing radio engines which are not powered by hardware acceleration can be performed on the same CPU device.

4.1.1 OpenCL Program Execution Mechanisms

Despite the detailed guide to OpenCL APIs, the OpenCL program can be split into two parts from the top, which are the host program and the kernel. The host is generally the same as other CPU programs which can do any desired jobs using various languages. The kernel is a scripted function that is going to be massive paralleled. The language used by the script is a subset of C language together with the APIs provided from the OpenCL

library [125]. After all, the program will start with the host running on the CPU as a general program. The host is responsible for scheduling and offloading some of its calculation tasks to the kernel which will be finished on the OpenCL devices. An example of the relationships among the program, host, kernel and OpenCL device is shown in Figure 4-4.

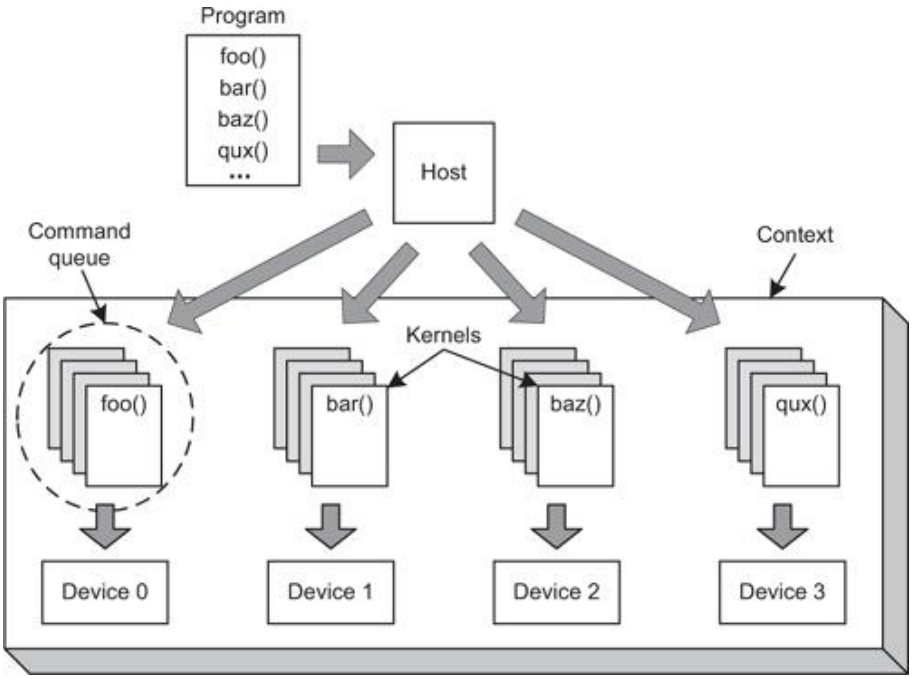


Figure 4-4 Kernels distributed from the host onto OpenCL devices [126]

Specifically, for the execution of each kernel from the host, a pipeline is given in Figure 4-5. Firstly, the chosen device needs to be initialized with the resource prepared for OpenCL jobs. The kernel script needs to be compiled into executable binary at the same time. Then based on the demand of kernel task, the corresponding context data needs to be pre-allocated on the device memory, including all the input and output of the kernel function. When everything is prepared, the actual context data will be mapped from the host memory to the pre-allocated device memory, followed by the execution of the kernel task. Once done, the host will download the output of the kernel from the device memory

back to the host memory. In the end, the allocated resource on the OpenCL device will be released.

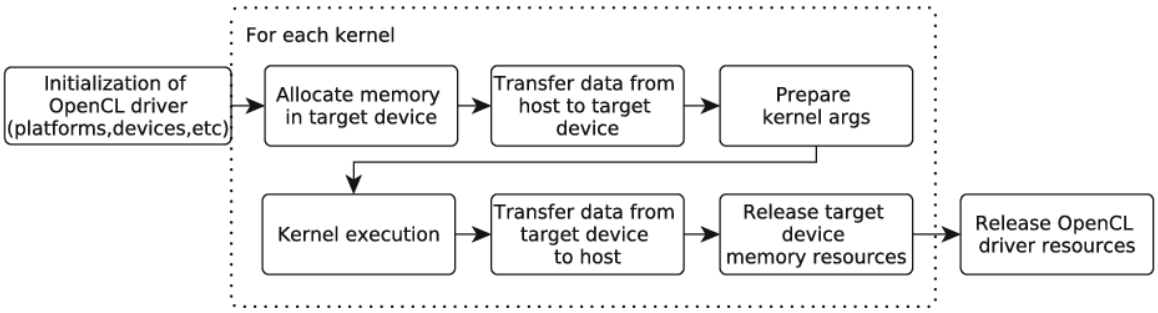


Figure 4-5 OpenCL kernel execution pipeline [127]

4.2 The Application of GO with the Design of the Ray Model

The GO provides the ray-based models with a mechanism to simulate the propagation of EM wave like lights. It includes a few light behaviours including propagation straight in homogenous medium, specular reflection and refraction [128]. It doesn't model the diffraction which is usually simulated by other methods like knife-edge or UTD. While for some indoor propagation models, the diffraction is not modelled as a matter of simplification and that reflection is dominating the contribution [58]. Some outdoor models also use an empirical approach for simulating diffraction to reduce complexity [11, 21]. For the model developed in this chapter, the diffraction is not considered, as the main purpose is the prototyping of a hardware-accelerated ray-launching model to understand the potential from hardware acceleration and difficulty in solving angular dispersion. The simulation of diffraction will be added in the next chapter. For the rest of the thesis, the model developed in this chapter will be denoted as GORL.

4.2.1 Propagation and Free Space Loss

A radio ray emitted from the transmitter travelling in the space can be modelled with an origin and a direction

$$\boldsymbol{r} = \boldsymbol{p} + t\hat{\boldsymbol{d}}, \tag{4-1}$$

where $\boldsymbol{p}$ is the location of the transmitter, $\hat{\boldsymbol{d}}$ is the normalized vector for direction, and t is the travelled distance in meter.

Then free space loss represents the loss between isotropic radiators in free space [129], which is given as

$$L_{FS} = (\frac{4\pi t}{\lambda})^2 \tag{4-2}$$

where λ is the wavelength in meter. Its logarithm expression is

$$\begin{aligned} L_{FS}(dB) &= 10\log_{10}\left(\frac{4\pi t}{\lambda}\right)^2 \\ &= 20\log_{10}(t) + 20\log_{10}(\frac{4\pi}{\lambda}) \end{aligned} \tag{4-3}$$

In total, the received power in decibels after travelling t meters in the free space will be

$$P_{rx} = P_{tx} + G_{tx} + G_{rx} + L_{FS}(dB) \tag{4-4}$$

where P_{rx} is the received power, P_{tx} is the transmitted power, G_{tx} is the transmission gain, G_{rx} is the reception gain. It should be noted that G_{rx} will be always given as 0dB in this study. As the holding of the handset is unpredictable from the mobile user, the reception antenna will be simplified as an isotropic antenna. Then the computation part in equation (4-4) will be left with $G_{tx} + L_{FS}(dB)$. For the GORL implemented in this chapter, an isotropic transmission antenna will also be assumed. Therefore, the concern of GORL will be left with $L_{FS}(dB)$.

4.2.2 The Modelling of Transmission and Reflection

According to Snell's Law [128], the light direction will be bent when transmitting through the area with a changing index of refraction, which is the same for EM waves. Given a homogenous wall with front and back surfaces in parallel, the ray exiting the wall will be in the same direction as the incident ray, but with a shift. The GORL neglects the shift through wall, and simplify the refraction into a transmission, i.e. the exiting ray will be in line with the incident ray. This is due to the limited building information in urban scenarios, that it is hard and cosy to acquire the thickness for each wall in practice [2].

For reflection, given a specular surface, the incident ray and reflected ray will be on the same plane. The incident angle will be equal to the reflected angle. To perform the reflection in vector, firstly mirror the incident ray origin direction against the reflection plane. Given the normal of the reflection plane represented as

$$\boldsymbol{r}_{normal} = \boldsymbol{p}_{normal} + t\hat{\boldsymbol{d}}_{normal}, \tag{4-5}$$

where $\boldsymbol{p}_{normal}$ is a point on the plane and $\hat{\boldsymbol{d}}_{normal}$ is the normalized vector of the plane normal. Then the mirror $\boldsymbol{p}'$ of the incident ray origin $\boldsymbol{p}$ is

$$\boldsymbol{p}' = \boldsymbol{v} - 2(\hat{\boldsymbol{d}}_{normal} \cdot \boldsymbol{v})\hat{\boldsymbol{d}}_{normal} \tag{4-6}$$

where

$$\boldsymbol{v} = \boldsymbol{p}_{normal} - \boldsymbol{p} \tag{4-7}$$

Denote $\boldsymbol{p}_{hit}$ as the hitting point of the incident ray on the surface, then the reflected ray will be

$$\boldsymbol{r}_r = \boldsymbol{p}_{hit} + t\hat{\boldsymbol{d}}_{\boldsymbol{r}}, \tag{4-8}$$

where

$$\hat{\boldsymbol{d}}_{\boldsymbol{r}} = \frac{\boldsymbol{p}_{hit} - \boldsymbol{p}'}{\|\boldsymbol{p}_{hit} - \boldsymbol{p}'\|} \tag{4-9}$$

Apart from the GO modelling of the ray during a reflection, the corresponding EM properties and interaction loss are be conducted by the formulas supplied in Section 2.2.2. The material database measured in the WIFEEB project D1.1 [130] gives the EM properties for typical building materials at different frequencies.

4.2.3 Ray Model

To carry the necessary information required above in the program, a Unified Modelling Language (UML) representation of the ray model class diagram is given in Figure 4-6. The type `RayWithPayload` is composed by a `Ray` and a `RayPayload`. The type `Ray` represents the geometrical part of the ray model, which has two fields, the `Origin` and `Direction`, which are both of type `Vector3F`. All the rest of the ray data required by the program are stored as `RayPayload`. It should be noted that the separation of a pure geometrical type `Ray` is for the efficiency of the OpenCL kernel. Since 90% of the calculation is spent in the ray intersection test [90], the kernel is designed to only offload the intersection calculation. Therefore, the minimal information required by the kernel function should be provided, to save the device memory and time used for uploading/downloading kernel context.

On the `RayPayload`, the counters of interactions are also stored as `NumberOfTransmission` and `NumberOfReflection`. This is to apply a termination of the ray once a maximum number of interactions is achieved, as a strategy used in other models as well [1, 90, 106]. By default, the thresholds are 5 for transmission and reflection.

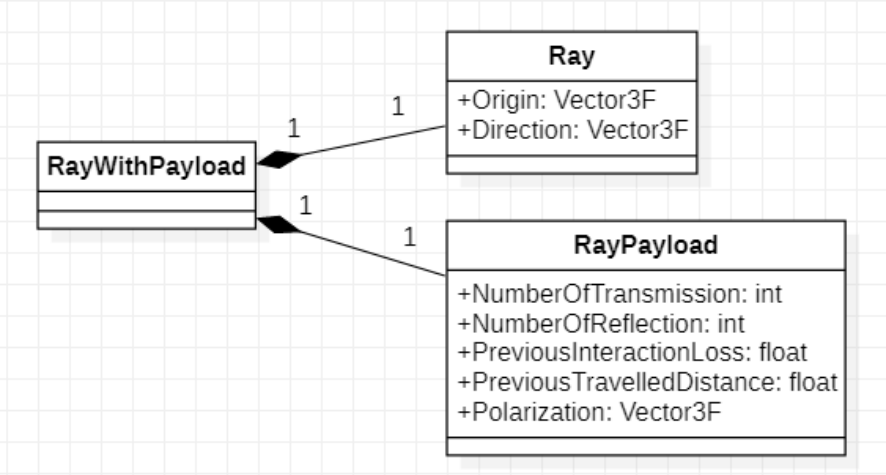


Figure 4-6 Ray model in UML diagram

4.3 Environment Modelling

The complex environment can be composed of the basic unit, polygon, which is a collection of vertices on the same plane. Similar to the ray, the polygon model is split into the geometrical part and the additional payload part, as shown in Figure 4-7. The separation of the geometrical part is for the kernel efficiency as it needs to be uploaded to the kernel buffer. The `PolygonPayload` stores the reference to a Material by its `MaterialId`. This is due to that many polygons may be of the same material. Then a particular `Material` instance can be fetched through the central storage `MaterialRepository` by calling the method `GetMaterialById(id)`. It should be noted that the GORL is a narrow band model, thus the material properties are supplied for a particular frequency once at a time.

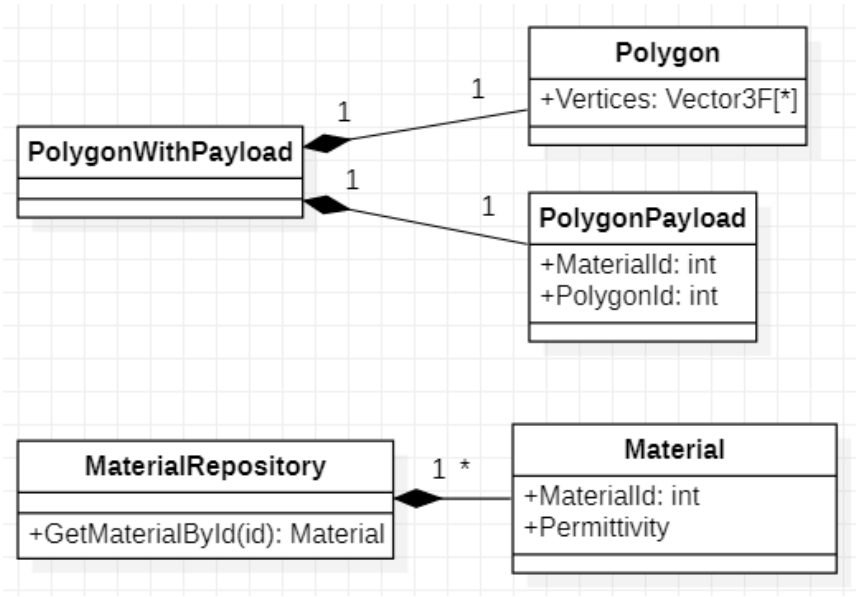


Figure 4-7 Polygon model and material repository

In the end, the on-site physical structures, e.g. terrain, external walls, grounds, plasterboards, windows and doors etc., can all be represented using the same type `PolygonWithPayload`, which is associated with a material. One special case is the terrain, which can be treated with infinite thickness. Therefore, it can be approximated by special material with an infinite loss to suppress the transmission, as the signal under the terrain is of no interest. It is worth mentioning that the terrain data are usually provided by standard GIS format. The import and modelling of terrain will be considered in the next chapter, but not in GORL.

4.4 The GORL Modules and Processing Pipeline

The GORL is composed of four modules, which are the four circles shown in the data flow diagram in Figure 4-8. The Initial Ray Generator is responsible for converting the antenna information provided by the client to initial pending rays. Then the Intersection Processor will detect the hit or miss of the pending rays on the scenario polygons. Once

the hit is detected, the processed ray will be sampled into pathloss results on voxels in the Ray Collector. In the end, the finished rays will be passed to the New Ray Generator, where the new rays from transmission and reflection will be generated and fed back into the pending rays. The green loop denotes the circulation from the intersection to collection, and to new ray generation, which is the core thread of the GORL.

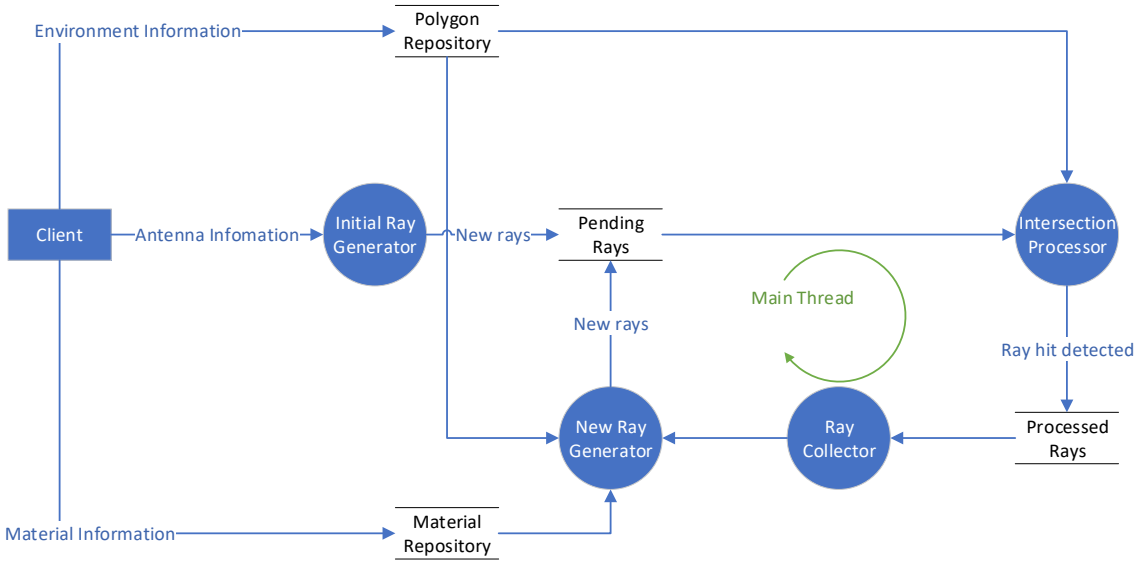


Figure 4-8 GORL modules data flow diagram

Specifically, for the Intersection Processor and Ray Collector, two types of implementations are provided. One schedules the corresponding task to be performed on OpenCL kernels. Another one is a replicate of the function but implemented on CPU host. Therefore, a straightforward comparison for the boost provided by OpenCL is available by switching the implementations for these two modules.

4.5 Intersection Detection and Triangulation

The intersection test is the key task in the propagation model. However, it is rarely discussed in the literature for ray-based radio model. Most of the studies described the

mechanism based on polygons. But in computer graphics, the triangle is the most used primitive rather than the polygon, due to its simplicity, safety and versatility [131]. On the one hand, the polygon with more than 3 vertices may have their coordinates not on the same plane due to floating error or data error. Such error would cause unexpected behaviour in the kernel. On the other hand, the triangle is a flexible shape that all the other primitives can be approximated with. Therefore, the GORL uses triangle as the basic element for ray intersection test. Many ray-triangle intersection methods are proposed in the literature. According to a benchmark comparison in [131], the method shown with the best performance in GPU is created by Möller [132], which is the used method in GORL.

The basic idea of Möller's method is by forming a coordinate system by two sides of the triangle, and a third axis from the vertex between the two sides, to the origin of the tested ray. Then the hit position can be solved by transforming the ray equation into the triangle coordinate. The advantage of this method is that no intermediate data need to be cached which is efficient for the limited memory on the kernel.

Meanwhile, since triangles are used in the kernel, the conversion from polygon to triangles is needed in the program. The triangulation is another research topic on its own. For this study, the speed of triangulation is less a concern as long as the polygon is parsed correctly. This is due to that the triangulation is a pre-processing stage of the scenario. On the one hand, the processed triangles can be reused for the rest of the antennas in the same scenario. On the other hand, the time spent on triangulation can be extracted from the analysis as it is not the focus of the study. After all, an open-source library PolyPartition [133] is selected which supports convex partitioning and triangulation using the ear clipping method [134].

4.6 Spatial Indexing Acceleration

Another issue of the ray-launching method is the complexity of determining the closest intersection for a ray in a complex environment, given all the triangles could be candidates, as discussed in section 3.1.3.2. The solution is by exploiting the spatial correlation among the triangles and narrow down the searching scope based on the geometrical properties of the ray. There are many mature acceleration structures used in graphics including BVH, uniform grid and octree, etc. [135]. The fundamental mechanisms behind these structures can be categorized into two types. The BVH like approach uses the bounds of the triangles and forms nodes which can be composed by a few neighbour bounds. Therefore, one triangle can only appear in one leaf node. This avoids the repetitive testing on the same triangle. The uniform grid like approach divides the space into cubes, in which the presence of triangles is calculated and cached. Therefore, one triangle can appear in multiple cubes and may be tested multiple times. But the topology of the uniform grid is better aligned with the ray propagation path. If the cubes have multiple levels and different sizes, then the structure becomes octree.

Comprehensive comparison among these structures is presented in [136], where the BVH was shown to have the top efficiency, with more than 80% relative speed to the fastest method while only consumes about 10% of the memory used by the fastest method. Therefore, this study uses BVH as the acceleration structure. A widely used BVH implementation is given in [137] where a binary BVH with axis-aligned bounding boxes(AABB) is used.

The BVH is a binary tree like structure as shown in Figure 4-9. Each node represents a cuboid volume aligned by the axis, given AABB applied. An intermediate

node will be composed of two sub-nodes dividing the parent volume. The bottom of the tree lays the leaf nodes, which contains several triangles in a cuboid. The BVH has a definition of maximum depth, which is the number of layers the tree can have, and a definition of maximum leaf size, which is the threshold of triangles counted in a leaf node to terminate the node division.

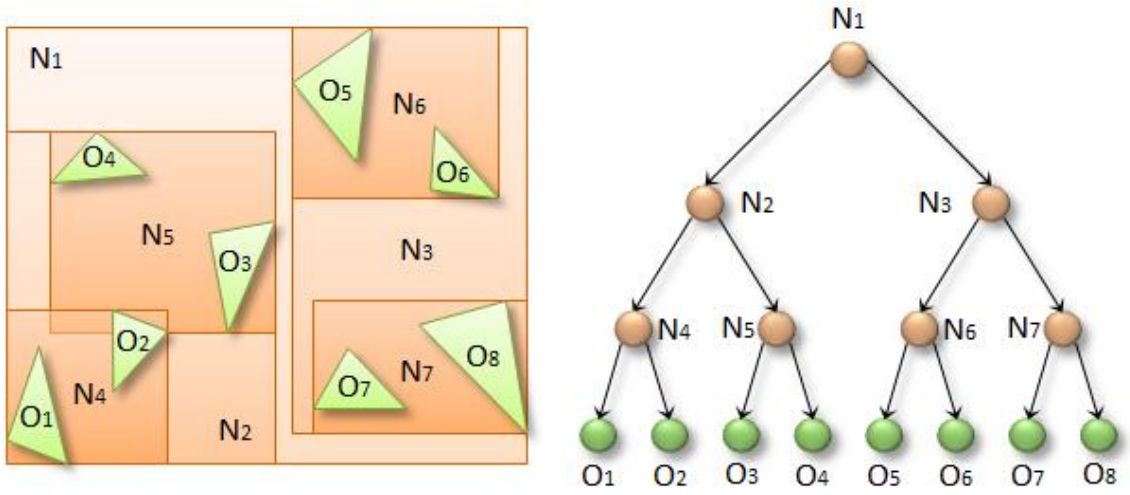


Figure 4-9 BVH tree structure [138]

To build the BVH, starting with the full scenario as the base node, firstly, compute the cuboid and identify the longest dimension. Secondly, sort all the N triangles by their centres along the longest dimension. Thirdly, form two child nodes with one by the first half of sorted triangles and another by the second half. Then repeat the process for the two child nodes. The generation will come to a stop once the maximum depth is reached, or the current node is having a number of triangles equal to or less than the given leaf node size. Ideally, the BVH should be defined big enough to include all the triangles without triggering the maximum depth condition. In this case, for a BVH with D levels, the planned maximum number of triangles is

$$Num_{maxTri} = Num_{leaf} \times Size_{leaf}, \quad (4\text{-}10)$$

where $Size_{leaf}$ is the leaf node size, and Num_{leaf} is the total number of leaf nodes which is given as

$$Num_{leaf} = 2^{D-1} \quad (4\text{-}11)$$

To travel through the BVH, the depth-first search (DFS) algorithm is used, i.e. the tree will be visited by one branch to the leaf node firstly before exploited horizontally. A typical implementation of the DFS algorithm is by a recursive function, i.e. a function calling itself in its body. However, it should be noted that the recursive function is not supported in OpenCL kernel [125]. Therefore, a stack, which is a first in last out data structure, is used to achieve the DFS. A pseudocode representation of the traversal function is given in Code Snippet 4-1, and the full kernel code of the BVH traversal is given in Appendix I GORL OpenCL Kernels. There are two kinds of intersection tests performed in the function. One is the ray triangle intersection test, for which Möller's method is used as introduced in section 4.5. The other is the ray box intersection test for the bound of each intermediate node.

```
function BVHTraversal(Node, Ray)
    let S be a stack
    S.push(Node)
    while S is not empty do
        Node = S.pop()
        if Node is not leaf node then
            if Ray hits Node then
                for each childnode in Node
                    S.push(childnode)
        else
            if Ray hits triangles in the Node then
                return Node
```

Code Snippet 4-1 Travel BVH by DFS algorithm using stack

There are only a few proposed methods for ray box intersection in the literature. The one widely adopted in many ray traversal methods is given by Smits [139]. Later, an improved method based on the Smits's was shown around 15% faster [140]. However, the improvement is achieved by caching three intermediate float values per ray for subsequent box intersection tests. The additional memory cost of 12 Bytes per ray is not GPU friendly, as the original ray type is composed of 6 floats which are 24 Bytes, as given in Figure 4-6. This means a 50% increase in the ray size. Given that ray is the major content that needs to be uploaded and cached in the OpenCL kernel, such an increase will sacrifice the upload speed and memory efficiency. Meanwhile, the test and conclusion in [140] were done on the CPU. Thus, the GORL uses the original Smits's method.

4.7 Grid Traversal and Ray Collection

Once the intersection of a ray is determined in the Intersection Processor, the status of the ray is finalized. Then the ray needs to be sampled from its origin to its termination

into pathloss results collected on a 3D uniform grid as shown in Figure 4-10. This involves the use of a ray grid traversal algorithm. A collection of grid traversal algorithms are surveyed in [141], and categorized into single layer traversal algorithms and hierarchical grid traversal algorithms, while the collection grid used in the study is a single-layer structure. However, there is no speed comparison provided for those methods, but only the method provided by Amanatides and Woo [142] is based on floating-point values. Given that the voxel size of the collection grid is from a user-defined collection resolution which could be any positive floating value, GORL adopts Amanatides and Woo's method.

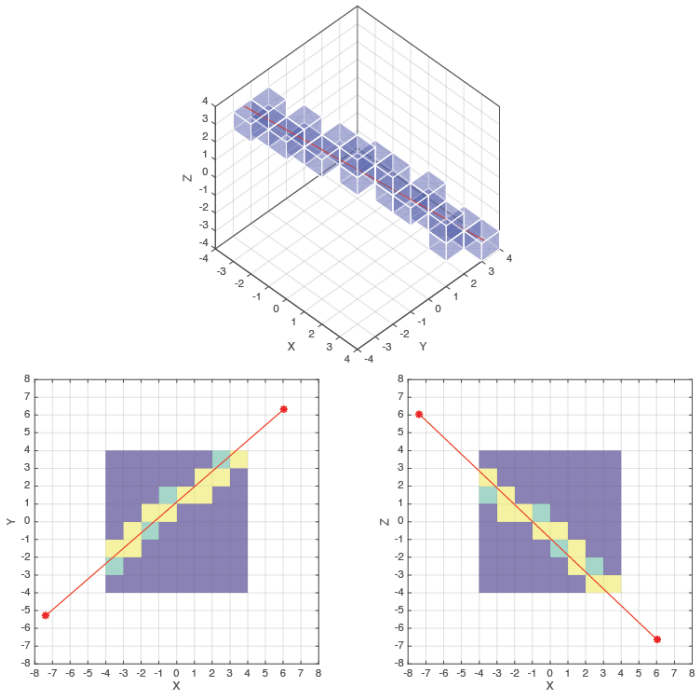


Figure 4-10 Ray traversal a the 3D grid [143]

Then by using the grid traversal method, for each voxel a ray walked into, the corresponding stored result needs to be updated with the pathloss carried by the ray at the location of the voxel. A 3D array of pathloss results for each voxel is contained in the

collection grid. In the end, the pathloss values can be rendered by a coloured legend and presented as heatmaps on different dimensions.

To calculate the free space loss, given an entered voxel i, by applying equation (4-3), the result at the voxel can be given as

$$L_{FS}(\boldsymbol{r_j}, i)(dB) = 20\log_{10}(t^i_{\boldsymbol{r_j}} + t^{pre}_{\boldsymbol{r_j}}) + 20\log_{10}(\frac{4\pi}{\lambda}) \quad (4\text{-}12)$$

where $\boldsymbol{r_j}$ is the collected ray, $t^i_{\boldsymbol{r_j}}$ is the travelled distance of the ray from its origin to its entry at voxel i, and $t^{pre}_{\boldsymbol{r_j}}$ is the accumulated length of all the parent rays that the current $\boldsymbol{r_j}$ is generated from. The value of $t^i_{\boldsymbol{r_j}}$ can be supplied directly from the grid traversal algorithm, thus the calculation of a vector's length is saved. The $t^{pre}_{\boldsymbol{r_j}}$ is the cached field `PreviousTravelledDistance` on the type `RayPayload` as shown in Figure 4-6. Therefore, whenever a new ray is generated from transmission or reflection, it will inherit the `PreviousTravelledDistance` from its parent ray, with the total length of the parent ray added at the same time.

In addition to the free space loss, the interaction loss that the ray encountered on previous hit polygon also needs to be accounted to get the final total loss. This is the cached field `PreviousInteractionLoss` on `RayPayload`. When a new ray is generated, it will inherit the `PreviousInteractionLoss` from its parent, with the current interaction loss added. Denoting the cached previous interaction loss as $L_{pre}(\boldsymbol{r_j})$, in the end, the total pathloss at voxel i can be given as:

$$L_{Total}(\boldsymbol{r_j}, i)(dB) = L_{FS}(\boldsymbol{r_j}, i)(dB) + L_{pre}(\boldsymbol{r_j}) \quad (4\text{-}13)$$

Lastly, multiples rays from many iterations may arrive at the same voxel, thus a combination strategy is needed. Generally, the pathloss results from different rays can be

combined by their weighted sum, average or maximum, depending on the reception facility. If a power delay profile (PDP) is measured, then the maximum can be used to predicate the strength of the main pulse in the PDP. For the purpose of cell planning, usually, a single value is required as an indication of coverage quality on a location. Ideally, the phases from different path should be considered to form the scale fading. However, due to the limitation of environment model detail and computation capacity, the phase is hard to be presented in a ray-based model explicitly. Therefore, stochastic approaches could be used to approximate the phase [52], or the mean value of different paths is used in some models [8].

However, it should be pointed out that different rays do not effectively mean different paths. This is the double-counting issue mentioned in Section 1.2. Therefore, the rays need to be distinguished into different paths and combined correspondingly. The demand for a path model will be introduced in the next chapter. For now, the GORL as a proof of concept will simply use the maximum value which does not have a concern for the path.

When the Ray Collector is implemented in OpenCL kernel, thousands of rays would be collected in parallel which may cause a race condition. For example, given a voxel storing a pathloss of 70dB already, ray A intends to update the voxel with 60dB while ray B intends to update the voxel with 50dB at the same time. If both rays send out the update commands, there is no guarantee which command will be the last one executed, therefore, the voxel now may randomly store a value of either 60dB or 50dB. From OpenCL 1.2, there are a set of atomic operations which would lock the memory access by one thread at a time [125]. The supplied `atomic_max` operation can be used to solve the race

condition above. However, this would cause a performance impact as some threads are now expected to wait for the finish of one thread. To ease the impact, the rays with less spatial correlation can be collected simultaneously, thus the wait is less likely to happen.

4.8 Ray Generation Strategies

The scheduling of the ray processing order also needs to be taken care of as a matter of memory efficiency. On hitting a polygon, child rays will be generated from a parent ray. Assume on average a parent ray can generate M child rays. For the case of GORL, the M equals 2 for a reflected ray and a transmitted ray. But it would be much larger if diffraction is modelled. Therefore, if the number of initial rays emitted from the transmitter is q_0, and the maximum number of interactions is N, then the number of rays at the Nth generation will be

$$q_N = q_0 \times M^N \tag{4-14}$$

Such exponentially increasing speed could use up the memory if all the Nth generation rays are created at the same time, especially when the q_0 is large in a medium-range scenario.

To cope with the issue, a DFS method could be applied, which is similar to the approach used in BVH node traversal in Section 4.6. A representation of the ray generations in a tree hierarchy is shown in Figure 4-11, where the big circle denotes a ray. Therefore, a strict DFS approach would only expand a single branch until the Nth generation before exploiting in the direction of breadth, and the maximum number of rays that would exist simultaneously is reduced to M. But this approach would cause a

limitation on the OpenCL kernel as a large number of rays is required to be processed in parallel to maximise the kernel performance.

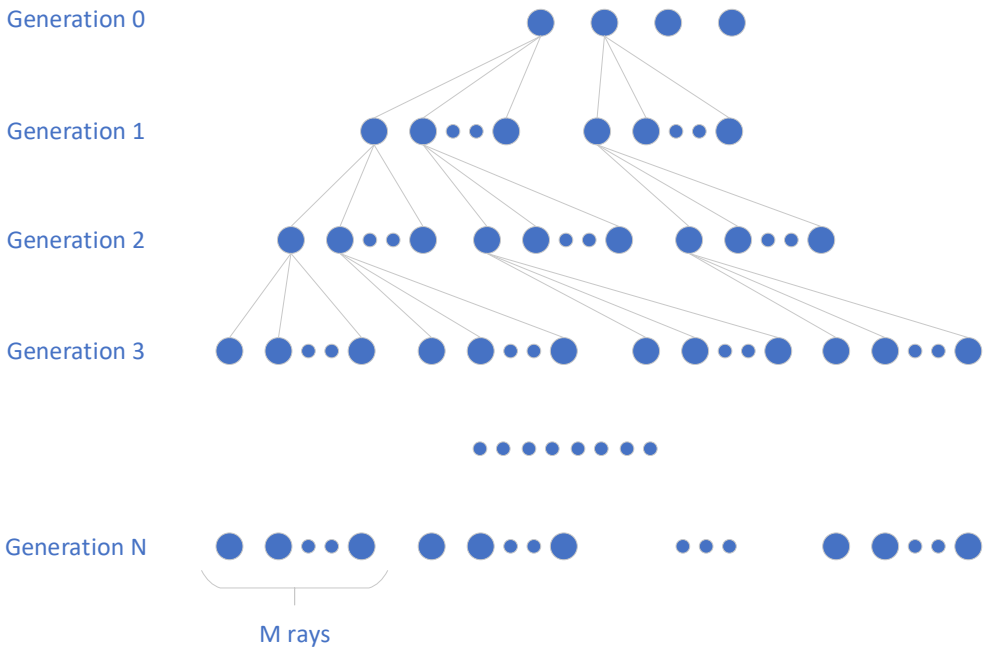


Figure 4-11 Ray generation hierarchy

Thus, a structure named priority stack is created to provide a DFS oriented order, while would not limit the kernel performance. The priority stack is a nested structure which is a stack of stacks. Each inner stack will be designated to a specific generation, and the stack for the Nth generation has the highest priority. When K rays are required from the priority stack for a kernel task, the structure will start with its highest inner stack to pop rays, and it will move to lower priority inner stack one after the other until the required number K is fulfilled. In this case, the priority stack will be DFS oriented as the Nth generation are always prioritised, and it won't block the kernel efficiency as a parent generation will start to be processed once a child generation runs out. Meanwhile, once a ray is generated in the New Ray Generator after an interaction, it will be added to the corresponding inner stack depending on the generation of the ray.

4.9 Simulation Result in an Outdoor Scenario with Medium Coverage

In this section, the implemented GORL model will be used for simulation in a medium-range outdoor scenario. The model is written using the language C#, and .Net Framework 4.6. There is a C# package wrapping the native OpenCL library, thus the OpenCL APIs can be accessed in C# easily. For the rest of the study, all the simulations will be performed on a machine with an Intel i7-8700 CPU (3.2GHz, 6 Cores), 16GB RAM and a GTX 1070 GPU with 8GB dedicated memory.

4.9.1 The Result and Efficiency Comparison of GORL on Difference Devices in the Paris Scenario

The used Paris scenario has a range of 1240m*1400m*50m with 39726 triangles in total. An omnidirectional antenna using 2.4GHz is placed at 7m above the ground with its location given in Figure 4-12. The simulations are calculated with resolutions at 5m and 2m. The results, speeds and memory consumptions are compared between the CPU and OpenCL implementation of the same algorithm. The GORL is also compared against the IRLA model.

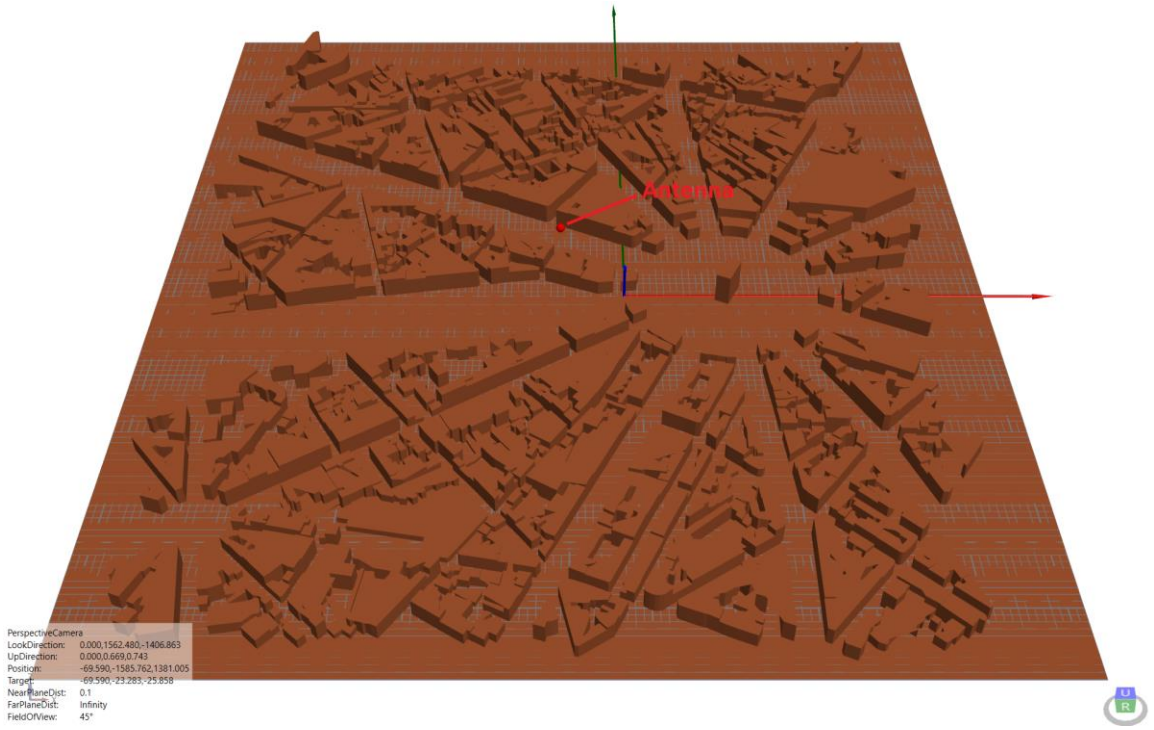


Figure 4-12 Paris scenario

The GORL result calculated by OpenCL is given in Figure 4-13. A reasonable coverage from the antenna is predicated. The effect of transmission and reflection with added pathloss can be observed. The validation of the result against measurement will be provided in the next chapter with a calibration module. For now, as a proof of concept, the presentation of the result is sufficient to demonstrate a workable solution. The rest of the discussion will focus on the comparison of efficiency.

	Time spent (sec)	Memory usage (MB)
GORL CPU	107.13	325
GORL OpenCL (CPU)	3.29	429
GORL OpenCL (GPU)	1.23	724

Table 4-1 GORL time and memory consumptions for Paris scenario at 5m

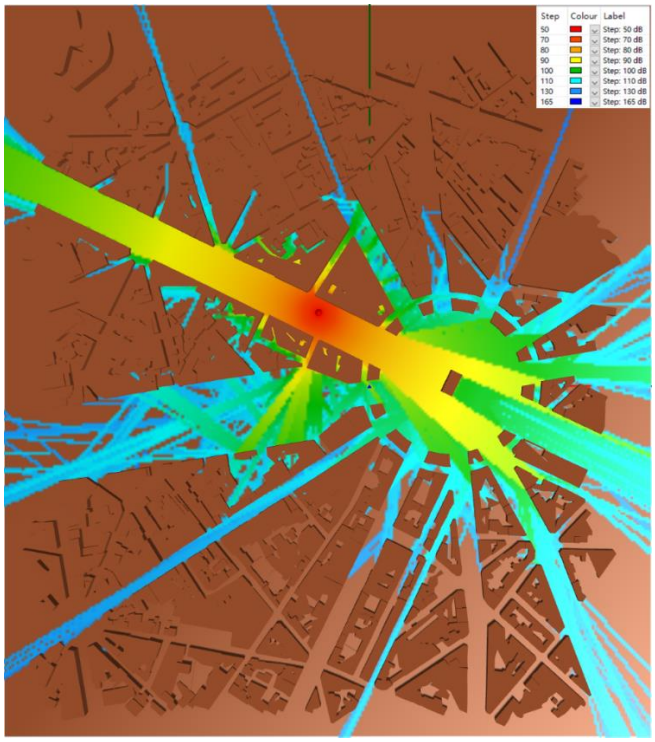


Figure 4-13 Paris GORL OpenCL (GPU) result at 5m

The time and memory consumptions of GORL are listed in Table 4-1. Three options are given in the comparison, where GORL CPU is the algorithm implemented locally on CPU, GORL OpenCL (CPU) is the OpenCL implementation with CPU chosen as the kernel device, and GORL OpenCL (GPU) is the OpenCL implementation running on GPU. Apart from the consumptions, the results of these options are the same, so they are not presented.

Firstly, the comparison between the speeds of CPU and OpenCL (CPU) shows a big improvement by 32.6 times on the same device with the use of OpenCL. This is proof of the feasibility of using OpenCL to overcome the complexity. The increased memory consumption from 325MB to 429MB is mainly due to the OpenCL buffer created to exchange the data between the host and the kernel.

Secondly, the comparison between the OpenCL (CPU) and OpenCL (GPU) shows further improvement in the speed by 2.7 times. The better speed from GPU may come from its stronger computing power as introduced in Figure 4-1. It could also be a result of offloading the kernel task to GPU and freeing the CPU so other host codes can be executed faster. On the other hand, the large increase of memory from 429MB to 724MB is due to the duplication of large chunks of data on the GPU device memory, which is not needed for the CPU kernel, as the host and kernel are on the same CPU device and can share the memory. After all, the GORL has presented a total speed improvement of 87 times which makes the proposal of using OpenCL feasible.

Then the calculation of the Paris scenario is repeated at resolution 2m, and the result is shown in Figure 4-14. Comparing to Figure 4-13, the presented plot from rays and pixels here are much finer as expected. However, the issue of angular dispersion is also becoming more obvious, as the separation of rays is getting clear after a few reflections. A brute force solution will be attempted to overcome the angular dispersion in Section 4.9.3.

	Time spent (sec)	Memory usage (MB)
GORL CPU	799	694
GORL OpenCL (CPU)	10.01	1198
GORL OpenCL (GPU)	2.42	1527

Table 4-2 GORL time and memory consumptions for Paris scenario at 2m

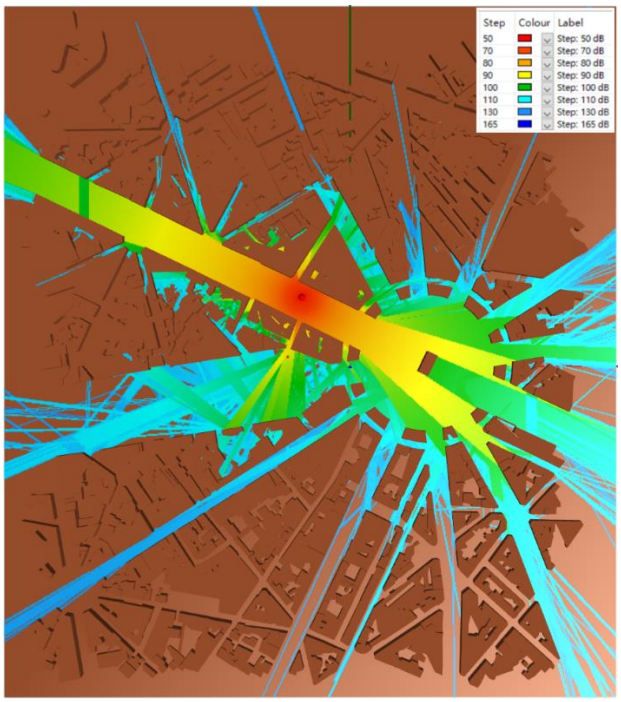


Figure 4-14 Paris GORL OpenCL (GPU) result at 2m

In the meantime, the consumptions at resolution 2m are given in Table 4-2. Comparing to Table 4-1, the time taken by CPU implementation is increased by 7.5 times. According to Equation (3-14), the complexity of ray-launching has a dependency on the length l of the scenario by l^2, which is effectively the inversion of the resolution given the same scenario, i.e. the complexity can also be given as

$$Num_L(N) \in O(N^2 R^{-2}) \tag{4-15}$$

where R is the resolution. Therefore, the increase of complexity from 2m to 5m can be estimated as $(2/5)^{-2} = 6.25$ times. Thus, for the CPU implementation, the increase in time approximates the estimated increase in complexity. However, this is not the case for the OpenCL implementation. The time consumptions for the OpenCL (CPU) and OpenCL (GPU) are increased by 3.0 and 2.0 times respectively. Such mismatch may be due to that, for the OpenCL implementation, there is some overhead not related to the resolution, for

example, the host scheduling. Thus, the increased resolution will majorly increase the complexity of kernel task which is only occupying part of the time spent.

4.9.2 The Comparison Between GORL and IRLA in the Paris Scenario

The GORL is also compared with IRLA in the Paris scenario. As introduced in Section 3.2, the IRLA is a high-performance ray-launching model provided with a solution of the angular dispersion problem. The other major differences from GORL are that IRLA is a 2.5D engine and is raster-based. The result of IRLA at 5m is shown in Figure 4-15. Comparing with the GORL result, both models provide similar coverage area, while the IRLA result tends to be more pessimistic judging from the legend. The accuracy will be justified with calibration in the next chapter. Meanwhile, the IRLA result is not presenting angular dispersion effect after several reflections. This is showing that IRLA does provide an effective solution to the problem.

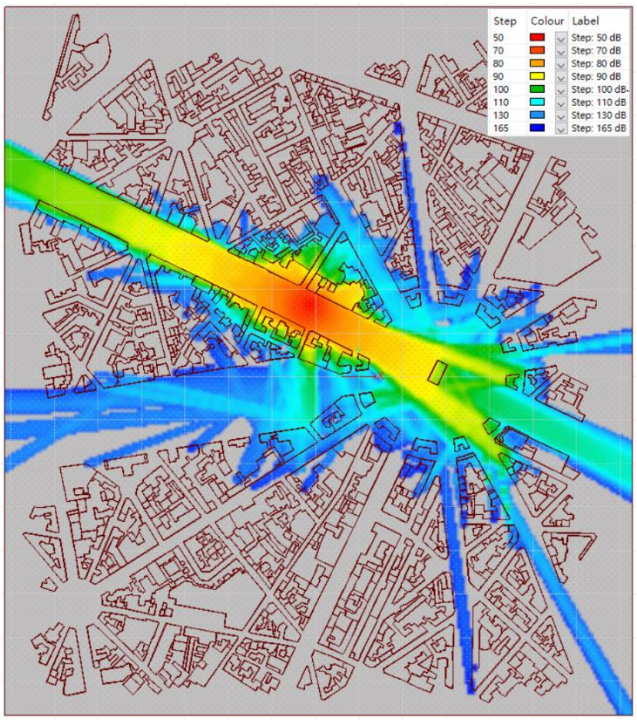


Figure 4-15 Paris IRLA result at 5m

In terms of the performance, the comparison of consumptions at different resolutions is given in Table 4-3. The status from GORL OpenCL (CPU) is used for a fair comparison on the same CPU device. In overall, the GORL is shown much faster than IRLA. At 5m resolution, GORL is twice the speed of IRLA, and at 2m resolution, it is 8 times faster. Memory-wise, the IRLA is more efficient than GORL. But if judging from the trend, the consumption of IRLA increases much faster by the resolution. As being a raster-based model, its algorithm complexity is linearly related to the number of voxels in the scenario. Given a fixed volume, the complexity can be given as

$$O(Num_{voxel}) = O\left(\frac{V}{R^3}\right) = O(R^{-3}) \tag{4-16}$$

where Num_{voxel} is the total number of voxels in the scenario grid, V is the fixed scenario volume, and R is the resolution of the voxel. Therefore, for IRLA, the complexity of 2m is about $(2/5)^{-3} = 15.625$ times the complexity of 5m. This scale approximates the ratio of time spent at 2m over 5m, which is 14.

After all, the comparison shows the advantage of the vector-based GORL over the raster-based IRLA, especially at high resolution. Not to mention the raster model won't be able to resolve multiple walls within a resolution, which is not the case for the vector model. These advantages are important for 5G prediction, where fine detail is demanded.

	Time spent (sec)	Memory usage (MB)
GORL OpenCL (CPU) – 5m	3.29	429
GORL OpenCL (CPU) – 2m	10.01	1198
IRLA – 5m	6.03	75
IRLA – 2m	84.1	486

Table 4-3 GORL compared with IRLA in Paris scenario at 5m and 2m

However, as mentioned above, the IRLA provides a solution for the angular dispersion which the GORL yet doesn't. A brute force solution in GORL is to increase the

density of initial rays to provide much finer angular resolution. According to Equation (3-14), the complexity has a dependency on the number of interactions N by N^2. Therefore, for a simulation with a maximum of 5 reflections as used above, the complexity can be increased by 25 times. Such an increase in complexity appears to be excessive for the current performance of GORL.

4.9.3 The Over launching Method and Result Discussion

To implement this brute force solution, in the Initial Ray Generator the rays are launched on the boundary voxels of a grid with a length of $11l$, where l is the length of the original grid. This method will be called over launching. Figure 4-16 presents the comparison of Paris results with and without over launching. On the left, some spots showing the angular dispersion effect are highlighted by red circles. It can be seen that such an effect is eased on the right result provided with over launching.

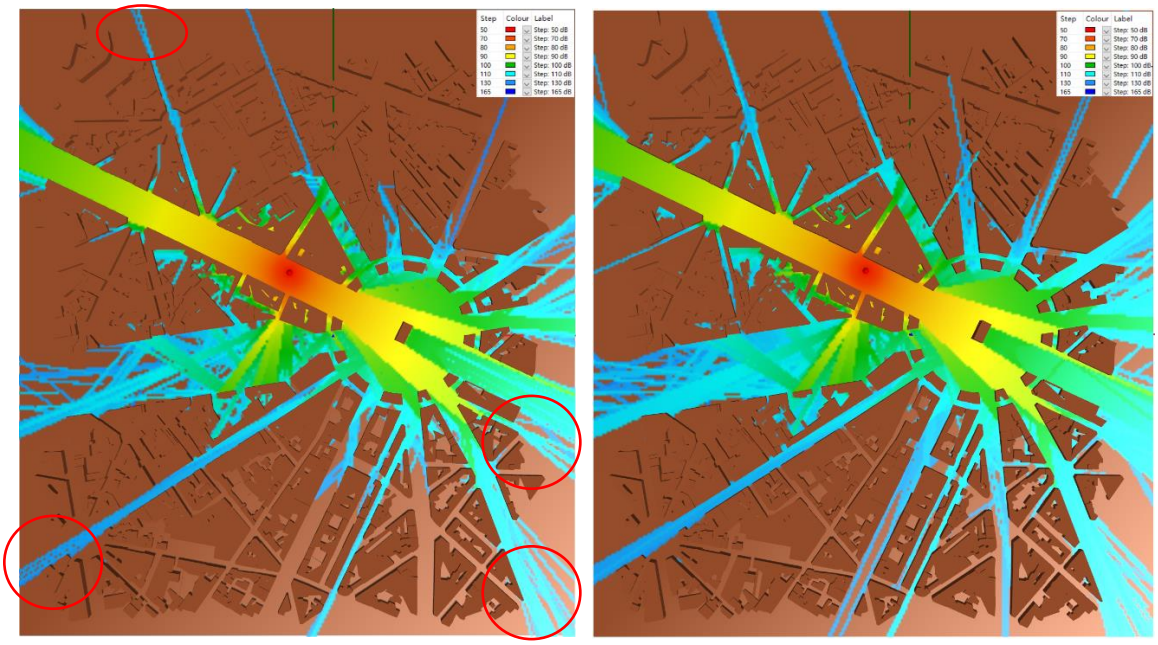


Figure 4-16 Paris 5m without over launching (left) vs. with over launching (right)

However, the increased consumption shows that the over launching is not a feasible solution for the angular dispersion. Firstly, with over launching, the time spent is increased from 3.29sec to 90.19sec, which is about 27.4 times slower. Secondly, the memory consumption is increased from 429MB to 3492MB. If the over launching method is used at 2m resolution, the engine failed with out of memory exception. Not to mention the Paris scenario is 1240m*1400m*50m, which is a small scenario given the desired target of providing medium-range prediction within a few kilometres.

4.9.4 The Optimal BVH Depth and Leaf Size

Put aside the angular dispersion problem, the speed of GORL is found with a dependency on the BVH depth, e.g. there exists an optimal BVH setting giving the best performance. Using the Paris scenario at 2m resolution, the simulation was repeated at different BVH depths, and the time spent was recorded for the OpenCL (CPU) and OpenCL (GPU) options. Each recorded time is the average of 5 duplicated runs. And the average leaf size is given by the total number of triangles, which is 39726, divided by the total number of leaf nodes at corresponding depth. As listed in Table 4-4, the result shows that the best speed is achieved with 13 depths for both CPU and GPU options.

BVH Depth	Average Leaf Size	GORL OpenCL (CPU) Time Spent(sec)	GORL OpenCL (GPU) Time Spent(sec)
8	310.36	26.45	5.88
9	155.18	17.46	4.22
10	77.59	13.57	3.35
11	38.79	11.14	2.77
12	19.40	10.08	2.48
13	9.70	10.01	2.42
14	4.85	10.08	2.97
15	2.42	12.04	4.29
16	1.21	19.76	11.97

Table 4-4 Time spent with different BVH depths in Paris scenario at 2m

The variation in speed is caused by different time spent on performing a bounding box intersection and on a ray triangle intersection. An estimation of the BVH traversal complexity is given as [137]

$$T = 2T_{AABB} + \frac{V(S_1)}{V(S)} Size(S_1) T_{tri} + \frac{V(S_2)}{V(S)} Size(S_2) T_{tri} \quad (4\text{-}17)$$

where T is the total time for a parent node traversal, T_{AABB} is the time for a bounding box intersection, T_{tri} is the time for a triangle intersection, S denotes the set of triangles contained by the parent node, $S_1$and S_2 are the sets for two child nodes, $V(S)$ is the volume of the S set triangles, and $Size(S_1)$ is the number of triangles in set S_1. The factor 2 before the T_{AABB} is for one box intersection on the parent node and another one on the child node. Specifically, for the BVH built in GORL, if $Size(S)$ is an even number, then $Size(S_1) = Size(S_2)$. Also, an approximation can be given as $V(S) \approx V(S_1) + V(S_2)$. Thus, Equation (4-17) can be approximated as

$$T \approx 2T_{AABB} + Size(S) T_{tri} \quad (4\text{-}18)$$

If iterate Equation (4-18) from BVH base node to the child node, then the total BVH traversal time can be estimated as

$$T_{BVH} = D * T_{AABB} + Size(Leaf) * T_{tri} \quad (4\text{-}19)$$

where T_{BVH} is the total BVH traversal time, D is the depth of the BVH and $Size(Leaf)$ is the average number of triangles per leaf node. Substituting Equation (4-11) into (4-19) gives

$$T_{BVH} = D * T_{AABB} + \frac{Size(S_0)}{2^{(D-1)}} * T_{tri} \quad (4\text{-}20)$$

where S_0 is the total set of triangles. The equation can be further simplified as

$$T_{BVH} = \alpha * D + \frac{\beta}{2^{(D-1)}} \tag{4-21}$$

where $\alpha = T_{AABB}$, and $\beta = Size(S_0) * T_{tri}$. Both α and β are constants. Therefore, the derivative of T_{BVH} with respect to D is given as

$$\frac{dT_{BVH}}{dD} = \alpha - \beta \ln 2 * 2^{1-D} \tag{4-22}$$

The derivative is monotonically increasing by D. Thus, the minimal T_{BVH} is achieved when the derivative equals 0. Then the optimal depth can be given as

$$D = 1 - \log_2(\frac{\alpha}{\beta \ln 2}) \tag{4-23}$$

It should be noted that $D \geq 1, D \in \boldsymbol{Z}$. Therefore Equation (4-23) may have no solution if

$$\alpha > \beta * \ln 2 \tag{4-24}$$

which can be converted as

$$T_{AABB} > \ln 2 * Size(S_0) * T_{tri} \tag{4-25}$$

In such case, it is an indication that the BVH shouldn't be used at all. However, the above condition can be rarely met in practical scenarios, where the $Size(S_0)$ would be big.

After all, Equation (4-23) gives an approximation for the optimal D value, but it is hard to be adopted in practice, as T_{AABB} and T_{tri} vary on different machines with no prior knowledge of their values. However, if assume that the ratio $r = T_{AABB}/T_{tri}$ can be approximated as a constant on different machines, then Equation (4-23) becomes

$$D = 1 - \log_2(\frac{r}{Size(S_0) * \ln 2}) \tag{4-26}$$

Applying the result in Table 4-4 with D equals 13 and $Size(S_0)$ equals 39726, then r can be estimated as $r \approx 6.72$. In the end, the optimal D can be estimated as

$$D \approx 1 - \log_2(\frac{9.69}{Size(S_0)}) \quad (4\text{-}27)$$

The result can be used as general guidance for an optimal BVH setting in a given scenario.

4.10 Conclusion

In this chapter, the principles of a GO-based ray-launching propagation model has been introduced. Then its implementation on the HPC platform OpenCL is given. Specifically, the detailed handling of the intersection test, spatial indexing acceleration and grid traversal etc. are investigated, which are rarely discussed in many other proposed propagation models.

Thereafter, in the simulation, the prototype propagation model GORL has shown great potential in providing the medium-range radio prediction. The GO-based vector method proved its advantage of less complexity dependency on the dimension or resolution of the scenario than the raster method. Also, the utilization of OpenCL has shown a significant boost of around a hundred times faster than a pure CPU implementation. And the calculation of a small cell can be finished within a few seconds. The magnitude of improvement and the final speed are much greater than another published result in [116] which uses the same Paris scenario with CUDA. The superiority of the achieved speed could be from the efficient choice of the intersection algorithm, spatial indexing method and neat ray model design.

However, it should be noted that the diffraction is not simulated in this model, as it is a prototype for investigating the hardware acceleration potential. Also as a solution to the angular dispersion, the over launching method is shown to be not feasible speed-wise nor memory-wise. This is due to the complexity dependency on the maximum number of

interactions N by N^2. Therefore, although the GORL does present the potential of achieving the speed target with hardware acceleration, it does not fully address the problem for providing the full 3D volume calculation. In the following study, the simulation of diffraction is added, and a modelling method is created to reduce the complexity $O(N^2)$ to a manageable level, to provide a final solution.

5 Tube Launching Model Based on Adaptive Ray Splitting

In this chapter, a tube launching model as a solution reducing the complexity dependency on the number of interactions will be proposed. In this model, the tube is approached by its central ray which would split adaptively to fill up the volume as the length of the tube grows. Therefore, the distance between two adjacent rays is guaranteed within one resolution regardless of the tube size. Thereafter, the modelling of diffraction will be introduced, followed by an improved module design and the calibration mechanism. Then, full implementation of the proposed propagation model will be given, with simulations and validations performed in a few scenarios. In the end, the limitations of the proposed method found during the implementation will also be discussed.

5.1 Continues Tube Model Approached by Ray Splitting

From the experience of GORL, it can be recognized that the fundamental cause of angular dispersion is due to the modelling of the continuous wave using discrete rays. The final sampling of rays into raster results is a compensation of the discontinuity. Enlightened by the idea, a solution to the root cause of angular dispersion may be sought by a type of continuous modelling methods. This thought leads to the few published tube-based methods [9, 10, 144, 145], as the tube is a continuous model with volume. Furthermore, to provide a feasible solution to the 3D volume radio prediction, the method needs to have the following 4 characteristics:

- The model shape can fully populate the 3D volume.

- The model can resolve environmental entities with a dimension larger than a given resolution threshold.
- Has an efficient algorithm for hit test between modelled shape and environmental entity.
- Has an efficient algorithm for raster result sampling from modelled shape.

The first requirement is to fulfil the continuous modelling as discussed above. The second one is to address the problem of angular dispersion. The third one is to make sure the method can provide an efficient simulation. The fourth one is related to the presentation of the result. As the raw model, either ray or tube, can't be intuitively used for analysis in radio prediction or planning, thus the sampling presenting the result in a different way is needed, e.g. heatmap as shown in Chapter 4.

The published methods mentioned above didn't cover the full 4 characteristics. In terms of the first condition, the tubes can be classified by their cross-section shapes, which are circle, triangle, square and hexagon etc. Among them, the circle cannot fully populate the 3D volume efficiently, unless overlap is introduced. For the rest, the square seems like a suitable option, given that it matches the cube shape of collection voxel. However, the rest characteristics are not met as discussed in Section 1.2. But the ray model used in GORL does provide a good fit for the rest conditions from 2 to 4.

Therefore, this study proposes a square tube-based method approached by rays. Unlike the methods in [10], where a single central ray or several edge rays are used to approximate the tube, the proposed method will use multiple rays to fill up the tube, with adaptive fitting. Starting with the transmitter, the initial tube will target the voxel on the grid boundary as shown in Figure 5-1, i.e. when the boundary voxel is reached, the cross-

section of the tube will be the same as the voxel. Such an initial tube can be effectively approached by a ray at the centre, as central ray will still be able to resolve entities larger than the resolution. The length of the central ray can be denoted by t_0, which is the distance from the transmitter to the centre of voxel on the boundary.

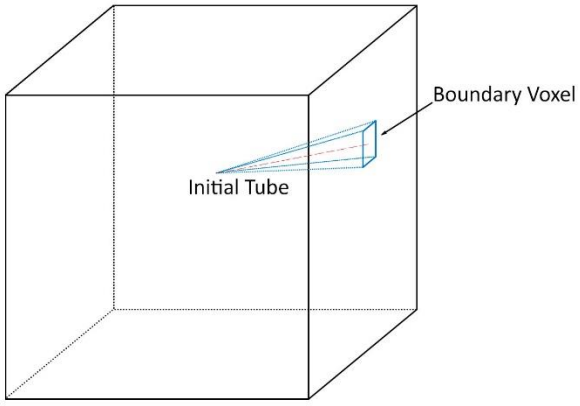


Figure 5-1 Initial tube targeting the boundary voxel

The problem happens when the tube is reflected in the scenario, therefore it would travel a longer distance than t_0 when the boundary is reached. Although the tube direction is changed after reflection, the problem can be simplified by modelling the reflected tube in the mirror which is still following the original direction. Thus, such an extended tube with a length larger than t_0 can be resolved by splitting the initial tube into 4 child ones when travels beyond t_0, as shown in Figure 5-2. When each child tube has travelled certain distance that its cross-section length equals the resolution again, the child tube can be split further.

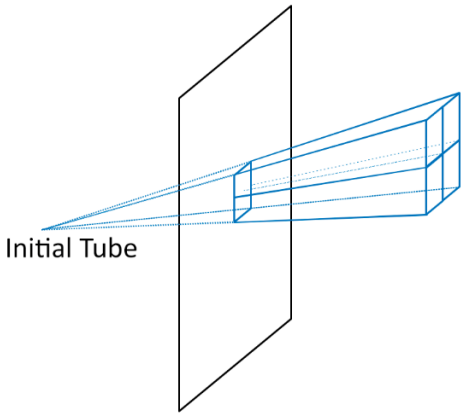


Figure 5-2 Tube splitting beyond one resolution

To model the tube and splitting using rays, one straight forward method is by using the central axis of each parent tube or child tube to form the ray, as shown on the top left in Figure 5-3. However, such method will result in a discontinuity in rays between a parent tube and its child tubes. If a parent tube ends right before hitting a triangle, then the central ray in the child tube has a chance starting behind the triangle without detecting the hit. Such bypassing may cause a room with low signal level to suddenly have an abnormal ray of high signal level as the transmission loss is missed. Another method that seems like the solution without bypassing is to start the central ray back from the source, as shown on the top right of Figure 5-3. However, after one reflection the source becomes virtual in the mirror for the reflected tubes. Therefore, the central ray from the virtual source needs an entry point into the real space, and an additional ray intersection test is needed for the point. A third method avoiding the overhead is by ray splitting as shown at the bottom of the figure. Each ray for a child tube, will start at the centre of the parent tube end, and aim at the centre of the child tube end. In this case, the transition from a parent ray to a child ray is continuous. Also, the ray starts in real space, so no additional hit test is needed.

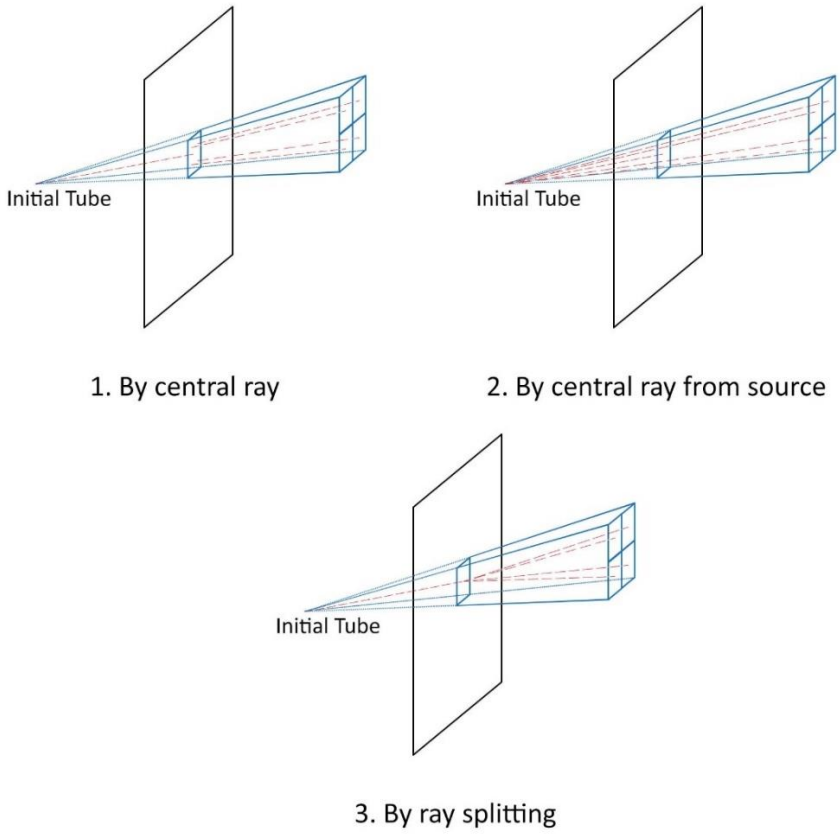


Figure 5-3 Methods modelling tubes by rays

It is necessary to make the tube self-contained, as several splits will make the tracking of all the parent tubes become overwhelming to a child tube and occupy the memory. Therefore, firstly the tube length needs to be derived. Given that one parent tube will be split into 4 child tubes, so the sum of the area of the child tube cross-sections is 4 times the area of the parent. Thus, the length of a child tube will be twice the length of the parent tube. Given an initial tube length t_0, then the nth child tube length t_n can be given as

$$t_n = 2^n * t_0 \tag{5-1}$$

Apart from the length, to derive a child tube, the source location and the plane equation of cross-section are needed. The source location can be carried as payload on the ray and is

be mirrored on every reflection. The plane equation can be expressed by two in-plane orthogonal vectors. They also need to be stored on the payload and mirrored. To help the ray splitting process, the vectors can be aligned to the coordinate axis when launching the initial tube. In the end, the child ray $\boldsymbol{r_n}$ can be given as:

$$\boldsymbol{r_n} = \boldsymbol{P_{end}}(\boldsymbol{r_{n-1}}) + t * \frac{\boldsymbol{P_{end}}(\boldsymbol{r_n}) - \boldsymbol{P_{end}}(\boldsymbol{r_{n-1}})}{\|\boldsymbol{P_{end}}(\boldsymbol{r_n}) - \boldsymbol{P_{end}}(\boldsymbol{r_{n-1}})\|} \tag{5-2}$$

where $\boldsymbol{P_{end}}(\boldsymbol{r_{n-1}})$ is the end of the parent ray $\boldsymbol{r_{n-1}}$ and $\boldsymbol{P_{end}}(\boldsymbol{r_n})$ is the end of the child ray which can be given as:

$$\boldsymbol{P_{end}}(\boldsymbol{r_n}) = 2 * (\boldsymbol{P_{end}}(\boldsymbol{r_{n-1}}) - \boldsymbol{P_0}) \pm \frac{R}{2}\widehat{\boldsymbol{v_1}} \pm \frac{R}{2}\widehat{\boldsymbol{v_2}} \tag{5-3}$$

where $\boldsymbol{P_0}$ is the source location, R is the resolution, $\widehat{\boldsymbol{v_1}}$ and $\widehat{\boldsymbol{v_2}}$ are the normalized axis-aligned in-plane vectors on the cross-section. The two $\pm$ operators will give the four ends for the 4 child rays.

In terms of the result sampling, the tubes are collected by corresponding rays similar to GORL. But the ray here is bent on each split, so its path cannot be directly used for free-space loss calculation in Equation (4-12). Instead, the actual path should be from the voxel centre to the source location, since now the source is mirrored and kept together with the ray. So, the ray itself will only be responsible for triggering voxels for collection using the grid traversal algorithm in GORL.

So far, the proposed method will be able to resolve entities beyond one resolution. Efficient intersection and result sampling are available since it is ray-based. Thus, the 4 conditions identified at the beginning of this section are likely to be met. For the rest of the study, the method will be named as TLARS which stands for tube launching adaptive ray splitting.

5.2 Multipath Modelling by Ray Batch

When combining multiple rays at the same voxel, the rays need to be distinguished by the path they come from, as mentioned in Section 4.7. Subsequent rays from the same path, e.g. from the initial source, should be omitted to avoid double-counting. Rays from different paths should be combined by chosen mechanisms. In [146], the double-counting issue is solved by the method of distributed wavefronts, which can be expressed using the following equation

$$\vec{E}(i) = \sum_{j=1}^{N} \vec{E}_j f(x_j) \tag{5-4}$$

where $\vec{E}(i)$ is the total collected field at voxel i, $\vec{E}_j$ is the field of jth ray collected on the voxel, x_j is the distance from the collected ray to the voxel centre and $f(x_j)$ is a weight function. For $x_j = 0$, $f(x_j)$ equals 1, and for $x_j = R/2$, $f(x_j)$ equals 0. Therefore, the weighted sum could compensate for the error from double-counting. However, this method does not distinguish the rays from double-counting and the rays between different paths, thus the contribution from multipath may be overwritten. In [78, 147], the double-counting is solved by introducing additional ray identification, i.e. a ray will be identified by a sequence of polygons it hit which could form a unique path. This method is adopted in TLARS.

To support the mechanism, TLARS introduces a new structure ray batch which is a collection of rays from the same path. Therefore, the transmitter would start with one batch containing all the initial rays. Then the initial batch could hit on M polygons, where for each of the polygon the hit rays could generate a batch of transmitted rays and a batch of reflected rays. So, the first interaction would generate $2M$ batches from the initial batch.

Thereafter, for the Nth interaction, the potential maximum number of batches could be $(2M)^N$. However, such a number will only be reached if each batch can hit all the polygons after every interaction, which is unlikely to be met. After all, the subsequent batches are all processed the same way as the initial batch, i.e. the rays from one batch will be divided into a few child batches based on the different polygons they hit. Thus, the ray batch structure can be used to maintain the unique identification of a path.

5.3 Diffraction

As for the diffraction, the UTD is the widely used theory in ray and tube-based methods [10, 148, 149]. An incident ray hitting the edge will generate a cone of diffracted rays, as shown in Figure 2-3. The incident angle between the incident ray and the edge equals the diffracted angle between a diffracted ray and the edge. The mechanism is applied to tubes when the incident tube hits on a segment along the edge, as shown in Figure 5-4.

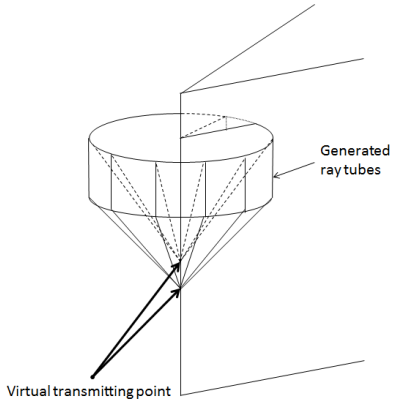


Figure 5-4 Incident tube hitting a segment along the edge generating the diffracted tubes [10]

However, these adoptions cannot be applied in TLARS directly, even the tube-based ones. This is due to that the type of tube shape is changed after diffraction.

Fundamentally, the tube model used in TLARS is from a point source. But a tube hitting an edge will inevitably hit a continuous segment. Thus, the source is becoming a segment instead of a point after diffraction. One way to solve the issue is to approach the segment emitted tube by ray splitting, which is similar to the modelling of point emitted tube. However, an attempt has found that the splitting rule would become complex with different splitting steps and cross-section vectors, due to that the convergence of the tube is different on different dimensions as shown in Figure 5-5. As during the profiling of the program, ray splitting is found disruptive in the engine processing pipeline, such complex splitting model would make it worse.

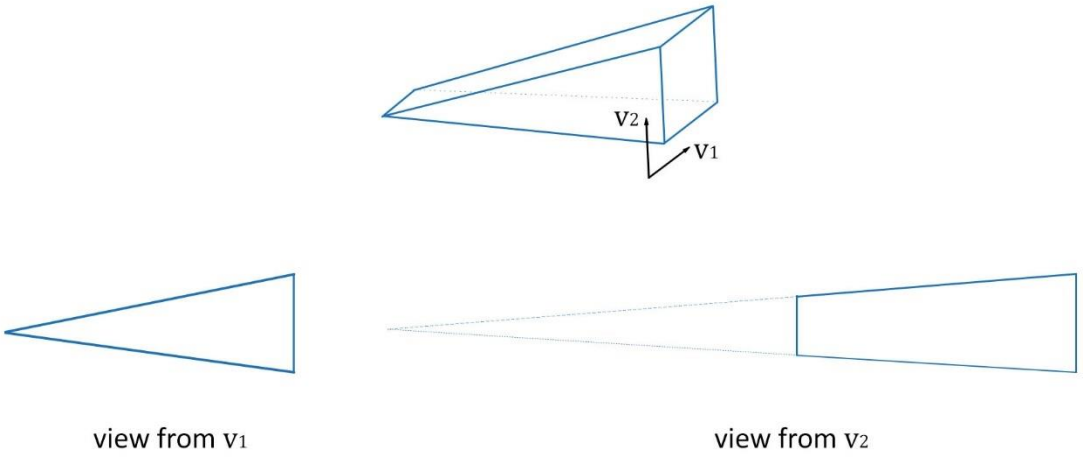


Figure 5-5 Segment emitted tube has different convergence on different dimensions

5.3.1 Segment Emitted Diffraction Tube Approached by Point Emitted Tube

Therefore, another way to model the segment emitted tube is to approximate it with a point emitted tube, which is used in TLARS. Assume a diffraction edge is laid on the xoy plane as shown in Figure 5-6, where an incident tube gets diffracted into tubes circling the diffraction edge. The first diffracted tube is in the same direction as the incident tube, it is created that it will split at the same location where the original incident tube will split. Then

the first diffracted tube will be rotated surrounding the diffraction edge into the shadowing area to create further diffracted tubes. It should be noted that each rotation will use the open-angle of the first diffracted tube so that there is guaranteed no gap between the diffracted tubes circling the edge. As the diffraction is creating new point sources, a large amount of complexity will be introduced. To ease the situation, the diffracted tubes circling into the opposite direction, i.e. the visible area, are omitted in the model.

If looking from the top in the horizontal view, a batch of incident tubes side-by-side would hit the diffraction edge as shown in Figure 5-7. Each incident tube will generate a diffracted tube which is the same as the first diffracted tube in the vertical view. Since the diffracted tube will split at the same location where the incident tube should split, it is guaranteed that there will be no gap horizontally as well. However, it can be noticed that the diffracted tube will start to overlap on the horizontal dimension after one split. This will result in redundant calculation in the overlapped area, which is an unavoidable drawback of approaching a segment source by point sources.

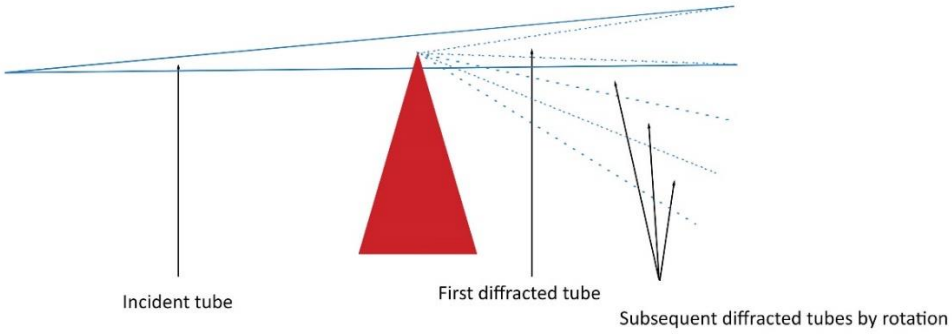


Figure 5-6 Tube diffraction vertical view

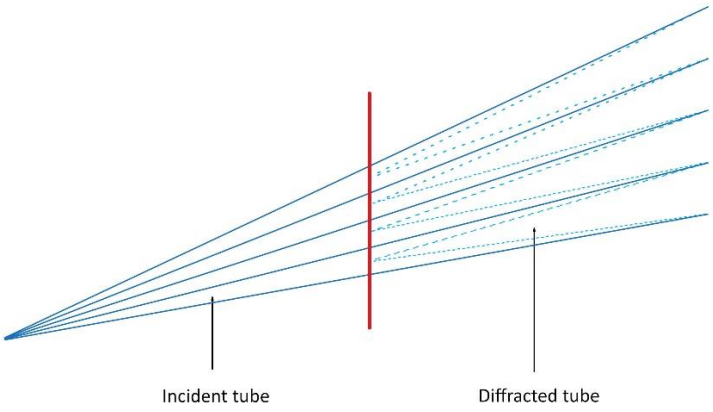


Figure 5-7 Tube diffraction horizontal view

5.3.2 Incident Ray Capturing

The method for how to trigger the diffraction on the edge by the rays also needs to be discussed, as the edge is a line, triggering by a direct hit on the line is not possible. This is a problem rarely mentioned in the literature. The solution provided by TLARS is to extend the diffraction edge into a polygon for capturing the bypassing rays. There are two ways to create the extension polygons. The first one is to use the neighbour walls of the diffraction edge as shown in Figure 5-8, where the top left corner of the block is the diffraction edge. On the top and left neighbour walls, two capturing polygons are created with a width of one resolution R. Therefore, an incident ray hitting the polygons will be denoted as a captured ray. Then, a shifted ray forming the actual diffraction path will be created from the source of the captured ray to a point on the edge, which is the projection of the captured ray hit. The shifting is a valid approximation, given that the ray is a representation of a tube, therefore, shifting within a resolution can be regarded as another approximated representation of the tube. In the end, the diffracted rays will be created in the shadowing area based on the shift ray. This method has its advantage of reusing the

neighbour walls for capturing. However, it will have limitation modelling the rooftop diffraction, e.g. the ray could be diffracted twice on the rooftop of a building on the front edge and the back edge. Such an effect has been verified and widely modelled in outdoor radio prediction [150, 151]. It is also stated as the dominating effect contributing to long-range urban coverage [152]. Thus, rooftop diffraction must be modelled in the model with a deterministic method.

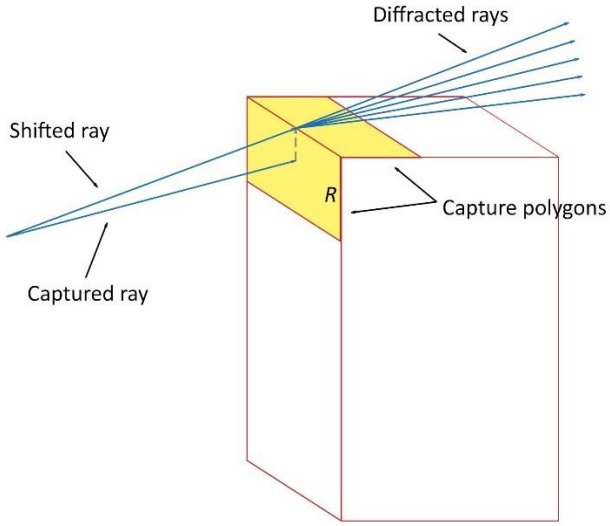


Figure 5-8 Capture incident ray by neighbour walls

Therefore, to support rooftop diffraction, a virtual diffraction polygon is created by reaching out from the edge, as shown in Figure 5-9. The virtual polygon is created with the same angle from both neighbour walls and has a width of one resolution. In this case, the ray bypassing the edge can now be captured by the polygon reaching out, then shifted and diffracted. Thus, the diffracted rays can also be captured by subsequent edge on the same root top, and the rooftop diffraction effect can be modelled. However, a drawback of this method is the termination of a ray on the virtual polygon, which doesn't exist in the real environment model. Thus, the terminated ray needs to be compensated by a new ray re-

launched from the point where the terminated ray was captured. So, a disruptive overhead has to be accepted as a full ray originally now is broken into two joint rays. The re-launched ray will be denoted as forward ray in the rest discussion.

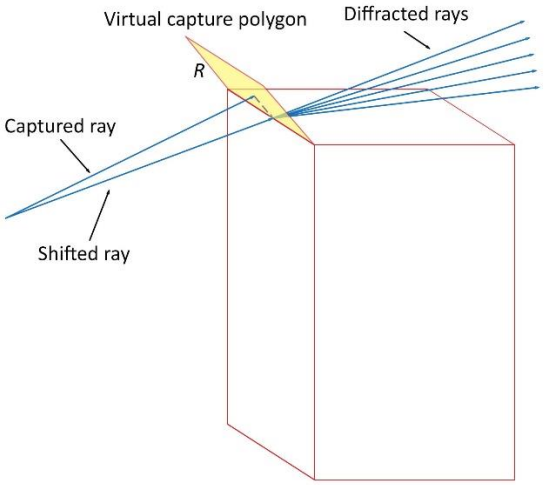


Figure 5-9 Capture incident ray by virtual polygon reaching out

5.3.3 Incident Rays Filtering

Rays from the same batch may hit the same virtual polygon, especially when the source of the batch is close to the polygon. This could create similar or identical shifted rays and end up with redundant diffracted rays generated. To improve efficiency, a filtering of the captured rays is introduced. As described in Section 5.2, rays from a batch are divided into hits on different polygons. Therefore, the full hits of rays from the same batch on a virtual polygon can be known in advance. Then the diffraction edge is divided into multiple segments with the same length. The incident rays from one batch are collected on each segment, and only the strongest ray per segment is shifted to trigger the diffraction.

On the other hand, the diffractions through a series of edges may trigger redundancy as well. As shown in Figure 5-10, assume a set of buildings have the same height and are in parallel, so the diffraction edges are at the same height. Then a batch B_0 coming from

the left will trigger diffraction on the first edge generating batch B_1. Then these two batches can both trigger diffraction on the second edge and generate batches B_2 and B_3. Thereafter, after the Nth edge, there could be 2^{N-1} incident batches generating 2^{N-1} diffracted batches. It can be seen that such exponential expansion is excessive, especially each diffracted batch can be approximated with a complexity of point source. Although the setup of the scenario is artificial, a similar situation is very likely to happen over the building tops, and along the streets with buildings aligned along the side. Meanwhile, the diffracted batch B_2 from B_0 would be much stronger than B_3 from B_1, as B_2 is having one less diffraction. But B_2 and B_3 are covering the same shadowing area. Therefore, to reduce the redundancy and the exponential complexity, a filtering rule is introduced. It is given as that a batch after diffraction can only trigger further diffraction if the ray has bent angle significant enough from the last diffraction, otherwise, a forward ray from the previous batch should cover the same area with stronger strength. Thus, in the scenario in Figure 5-10, at the Nth edge, there could be N incident batches and only 1 diffracted batch will be generated. So, the total number of batches after the edge is reduced from 2^N to $N + 1$.

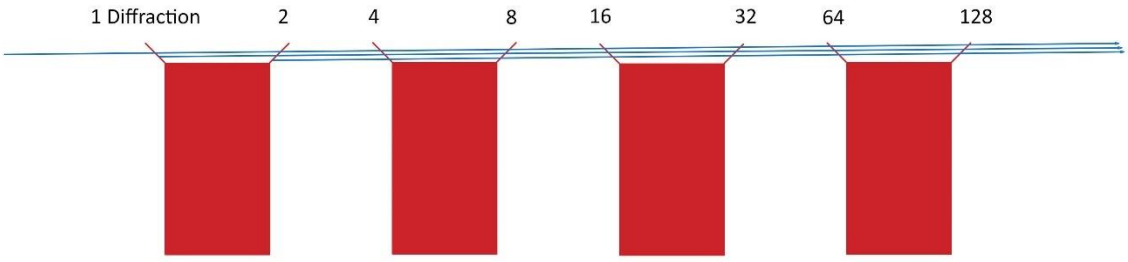


Figure 5-10 Number of diffractions expands exponentially through edges

5.4 The TLARS Module Design and Processing Pipeline

5.4.1 TLARS Ray Model

To support the behaviours introduced in the sections above including the tube modelling and diffraction, the TLARS has expanded its ray model with more fields in addition to the design given in Figure 4-6 for GORL. The TLARS ray model is given in Figure 5-11.

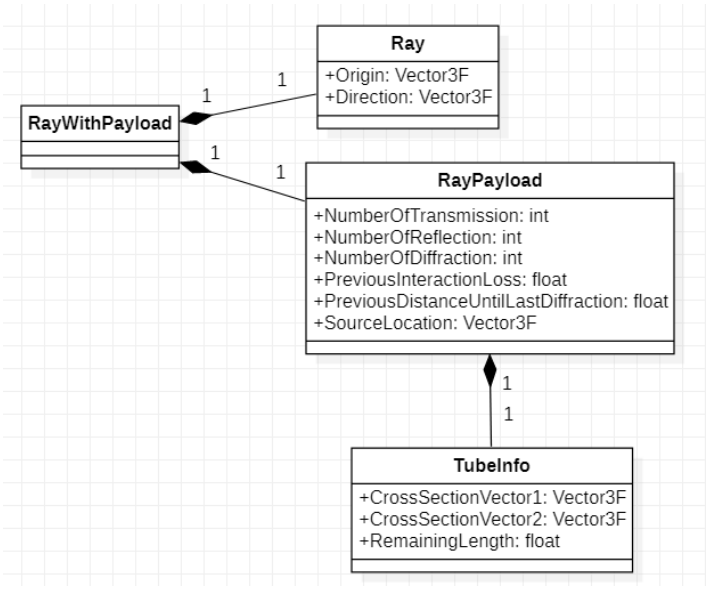


Figure 5-11 TLARS ray model in UML diagram

Firstly, to support the tube modelling, the structure `TubeInfo` is added which contains two fields `CrossSectionVector1` and `CrossSectionVector2` denoting the two in-plane vectors on the cross-section, as the $\widehat{\boldsymbol{v_1}}$ and $\widehat{\boldsymbol{v_2}}$ given in Equation (5-3). Another field `RemainingLength` storing the left distance the ray needs to travel before splitting is also needed. When a ray is generated from splitting, its end is provided by Equation (5-3). However, the reflection would change the direction of the ray so is its end. Thus, the length of the ray until its end is stored instead of the actual coordinates of the end. This could save

the mirror operation and memory. Then, if a polygon is hit, the generated transmission and reflection rays will carry on the `RemainingLength` deducing the length of the incident ray until its hit.

The check of splitting is triggered whenever a hit is detected. The length of a ray from its origin to the hit point will be compared with the field `RemaingLength`. If it is exceeded, then no new ray would be generated from the intersection, but the incident ray will be split by the `RemaingLength`. The intersection is expected to be tested again by the split rays with finer resolution. Therefore, all the polygons are guaranteed to be resolved within a resolution. The generation of the split ray is provided by the Equation (5-3).

Meanwhile, the field `SourceLocation` which represents the source, i.e. the tube origin, is added to the RayPayload. The `SourceLocation` will be mirrored against reflection polygon, as discussed in Section 5.1. This field will be used during ray pathloss collection to provide the total travelled distance so to calculate the free space loss given in equation (4-12). It should be noted that the ray length used in this equation is no longer applicable for the tube model, as a ray and its child rays may not be on the same line, given by the adaptive splitting mechanism described in Section 5.1. So, the free space distance should be the vector length from an entered voxel centre to the ray source location, which is given by the `SourceLocation`.

Then during diffraction, new point emitted tubes are generated from an incident tube. Thus, a new source is created regardless of where the incident tube is coming. Therefore, instead of carrying on the incident tube, the generated tube status is reset by the new point source on the edge, according to the mechanism given in Section 5.3.1. As a result, the tracking of total travelled distance is lost since the source is reset to a new point.

In order to supply the free space equation, the previously travelled distance until the last diffraction point, $t_{\boldsymbol{r_j}}^{pre-diff}$, is be cached in the `RayPayload` as `PreviousDistanceUntilLastDiffraction`. Therefore, the free equation (4-12) should be modified to

$$L_{FS}(\boldsymbol{r_j}, i)(dB) = 20\log_{10}(\left\|\boldsymbol{p_i} - \boldsymbol{p}_{\boldsymbol{r_j}}\right\| + t_{\boldsymbol{r_j}}^{pre-diff}) + 20\log_{10}(\frac{4\pi}{\lambda}) \quad (5\text{-}5)$$

where $\boldsymbol{p}_{\boldsymbol{r_j}}$ is the current source location of ray $\boldsymbol{r_j}$, and $\boldsymbol{p_i}$ is the centre of voxel i.

5.4.2 TLARS Modules with Maximized Parallelization

The TLARS engine has made some improvement based on the experience from the GORL engine. Firstly, the engine now is implemented using native language C++ 17 for better memory control than the managed languages C#. Secondly, the TLARS maximise the parallelization for different modules. In GORL, it was aware that, the host CPU, the intersection OpenCL kernel and the collection OpenCL kernel can be treated as different physical devices. Thus, there is no need for their tasks to be executed in sequence. In the meantime, the parallelization enables the use of multiple OpenCL devices simultaneously. This also gives an attempt to increase the performance regarding the number of rays traced per second. Thirdly, TLARS introduces the modelling of multipath by ray batch. Therefore, during collection, the ray from different paths can be combined using a chosen mechanism.

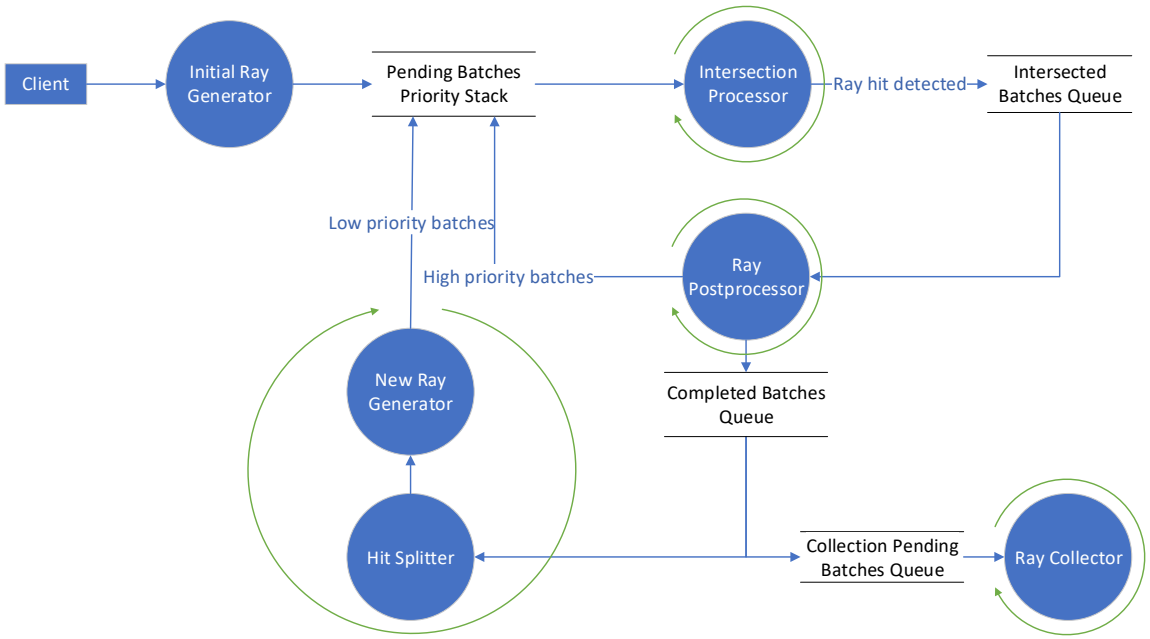


Figure 5-12 TLARS modules data flow diagram

The program design of TLARS is given in Figure 5-12. There are four main modules marked by the green circles, denoting that they are running in parallel on their own threads. In this case, the processing of OpenCL kernel tasks can be de-coupled from the complex CPU logics so to maximise the performance of the kernel. However, to achieve the parallelization, a drawback is that buffers need to be introduced to the boundaries of the modules so that once a module finishes its job, it can leave the result in the exit buffer to be picked up by the next module when that module becomes free. Otherwise, the prior module will have to wait for the next module to hand over the data. In the figure, four buffers, i.e. the queues and stack, are added to the design. It can be noticed that the entry and exit of each module are all linked to buffers.

Specifically for the Pending Batches Priority Stack, it is used to provide a DFS oriented processing order to minimise the concurrent rays for less memory consumption, as discussed in Section 4.8. The difference is that each node is a ray batch now in TLARS,

instead of a single ray in GORL. In addition, the Priority Stack introduces one more level of priority by having two internal stacks independently. The high priority one will be used for storing the incomplete batches from the Ray Postprocessor. The low priority one will be used for storing the newly generated batch from the New Ray Generator. Therefore, the incomplete batches will have a higher priority over the new batches when the Intersection Processor is pulling batches from the stack.

The whole pipeline starts at the Initial Ray Generator, where the ray tubes are generated from a given transmitter. They are stored as an initial batch in the Pending Batches Priority Stack. Then the Intersection Processor will run in its thread keep checking if there is any batch in the pending stack. Once found the batch will be processed with intersection tests and then stored in the Intersected Batches Queue.

Then the Ray Postprocessor will keep pulling ray batches from the Intersected Batches Queue. In there, the intersected rays will go through two checks. Firstly, each ray will be compared against its `RemaingLength` to see if splitting is needed. If so, the split child rays are generated in the batch, then the batch will be marked as incomplete and pushed back to Pending Batches Priority Stack with high priority. Secondly, if any ray hits a virtual diffraction polygon, a forward ray will be added to the batch which will also be pushed back to the stack with high priority. Therefore, these incomplete batches will be processed again by the Intersection Processor in its next cycle.

For the remaining complete batches after the Ray Postprocessor, they are firstly duplicated to the Collection Pending Batches Queue waiting to be collected, meanwhile, they are also sent to the New Ray Generator. The duplication of the batch in the queue would decouple these two modules, so the Ray Collector could operate in its own thread

and keep pulling from the queue. In the Ray Collector, the ray will be collected as pathloss values on voxels.

On the other hand, new batches are generated in the New Ray Generator from interactions, i.e. reflection, transmission and diffractions. However, before generation, the completed batch firstly needs to be divided into rays hitting on different polygons, which is done in the prior Hit Splitter. So that child batches, as modelling of different paths, can be generated from different polygons, as discussed in Section 5.2. In the end, these batches are pushed back to the Pending Batches Priority Stack with a low priority, awaiting to be intersected again.

After all, from the design, the four main modules are running in parallel to maximize their performance without dependency on the speed of the other modules. Each module will store its finished data in storage at the exit and will keep checking new tasks from storage at the entry.

5.5 Calibration

The calibration of building materials will be provided in TLARS. The material properties can be calibrated according to measurement with pathloss values on a collection of points. Similar to Section 3.2, the GA will be used to optimise the result, and the RMSE between the calculated result and the measured values will be used as the fitness equation in the GA.

5.5.1 The Cache and Replay of Ray History

In order to support the calibration, the core facility needed from the model is that the calculation needs to be repeated thousands of times to perform the fitness equation with

different material sets. This means the calibration will be thousands of times slower than a corresponding pathloss calculation in the same scenario. Such a scale would become not feasible for large scenarios.

However, the given measurement points usually won't cover all the locations in the scenario, especially in the z dimension. Therefore, the problem can be reduced to repeating the pathloss calculation on certain points. To do so, the rays arrived at those points need to be cached instead of being sampled and discarded in the Ray Collector. Meanwhile, to be able to reproduce the pathloss at those points, all the precedent rays and their corresponding interaction context need to be available as well, so that the pathloss can be replayed by tracking back to the original transmitter with different material sets. The full chain of rays from the transmitter to a measurement point will be denoted as a ray history.

But still, the caching of ray histories on measurement points is not a simple task. This is due to that now the cached ray can't be discarded in time to free the memory. Therefore, the strategy of the DFS algorithm is becoming of no use and the memory would be very likely filled up during the calculation, as discussed in Section 4.8. Moreover, the demand for interaction context, i.e. the knowledge of which material is interacted with what type of interaction, will also incur overhead in the memory consumption.

Thus, to overcome the problem of increased memory consumption, the scope of cached ray histories needs to be narrowed with prior knowledge of the useful rays among the full initial rays that could reach the measurement points. So, the calibration process is divided into three steps: firstly, the collection of useful rays; secondly, the collection of ray histories for the useful rays; thirdly, GA optimization by replaying the collected ray histories.

5.5.2 Collect Ray IDs for Effective Rays Hitting Measurement Point

If each initial ray is given with a unique ID, and each child ray inherits the same unique ID from its parent, then the set of useful initial rays can be known by collecting the IDs of the rays that arrived at the measurement points. To do so, the TLARS pipeline given in Figure 5-12 is reused as it is, apart from that the Ray Collector is replaced with the Ray ID Collector module.

However, given a ray batch, the number of rays that arrives at measurement points is random, which means the number of collected IDs is variable. This would cause a problem for OpenCL, as all the input and output buffer of OpenCL kernel must be fixed size per kernel call so that the dedicated OpenCL memory can be pre-allocated and secured by the hardware [125]. Therefore, another prior knowledge of how many rays that would arrive at measurement points is needed so that the allocation size of ray IDs can be known before calling the kernel. Thus, the Ray ID Collector is implemented into two OpenCL kernels, a Ray ID Counting Kernel followed by a Ray ID Collection Kernel.

In Ray ID Counting Kernel, given a few batches to be collected, the total number of rays arriving the measurement points is returned. Since the return value is a single integer, the OpenCL requirement of fixed size buffer is not violated. A simplified pseudocode representation of the counting kernel is given in Code Snippet 5-1, and the full kernel function is given in Appendix II TLARS OpenCL Kernels. The kernel still uses the Amanatides and Woo's method [142] for grid traversal which is the same as GORL. Besides, the buffers of `MeasurementPoints` and `Count` is introduced. When a voxel is entered in the grid traversal, its position will be checked with the `MeasurementPoints`. If

the voxel is contained, which means one ray has arrived at a measurement point, then the global counter `Count` will be incremented by 1.

```
function RayIDCountingKernel(Grid, Ray, MeasurementPoints, Count)
    for each Voxel in the Grid the Ray walks into
        if MeasurementPoints contains Voxel
            Count = Count + 1
```

Code Snippet 5-1 Ray ID counting kernel

Once the counting kernel is finished, the same set of ray batches will be repeated in the Ray ID Collection Kernel as given in Code Snippet 5-2, which is very similar to the counting kernel. The full code of the kernel is given in Appendix II TLARS OpenCL Kernels. The major difference is inside the inner loop. When a ray enters a measurement point, its ID will be captured by the buffer `CollectedIDs`. Now because the number of rays that would be captured is known from the above counting kernel, the buffer `CollectedIDs` can be pre-assigned with known size.

```
function RayIDCollectingKernel(Grid, Ray, MeasurementPoints,
CollectedIDs)
    for each Voxel in the Grid the Ray walks into
        if MeasurementPoints contains Voxel
            Insert Ray.ID into CollectedIDs
```

Code Snippet 5-2 Ray ID collection kernel

5.5.3 Collect Ray Histories for Effective Rays Hitting Measurement Point

Once the IDs of the useful rays are collected, these rays will be launched from transmitter again, and this time the ray histories will be collected based on the narrowed scope. The model of ray history is given in Figure 5-13. Each `RayHistory` is one full chain of rays arrived on a measurement point, which should also be located in the collection grid.

So, the `RayHistory` stores the measurement point as the field `CollectedVoxel`. Meanwhile, a `RayHistory` contains an `InteractionHistory`, in which a reference to its previous interaction is kept as the field `Predecessor`. Therefore, the `InteractionHistory` is forming a linked list, in which the interactions along the path can be tracked. Specifically, for one `InteractionHistory`, the `InteractionContext` is used to store the information during that interaction so that it can be played back. The `InteractionContext` is a discriminated union, which can hold the value of a few different types. In this case, the `InteractionContext` can be a `TransmissionContext`, `ReflectionContext` or `DiffractionContext`. According to the Equation (4-13), the full pathloss is given by the interaction loss plus the free space loss, and according to Equation (5-5), the free space loss calculation requires the source location, and previous distance until the last diffraction. Thus, for transmission, only the material is needed to provide the transmission loss, as the source location is not changed. While for reflection, the location of the mirrored source is needed in addition. Then for diffraction, the location of the diffracted point is needed since a new source is created from there. Meanwhile, the rotated angle during diffraction is needed as well to apply the diffraction loss.

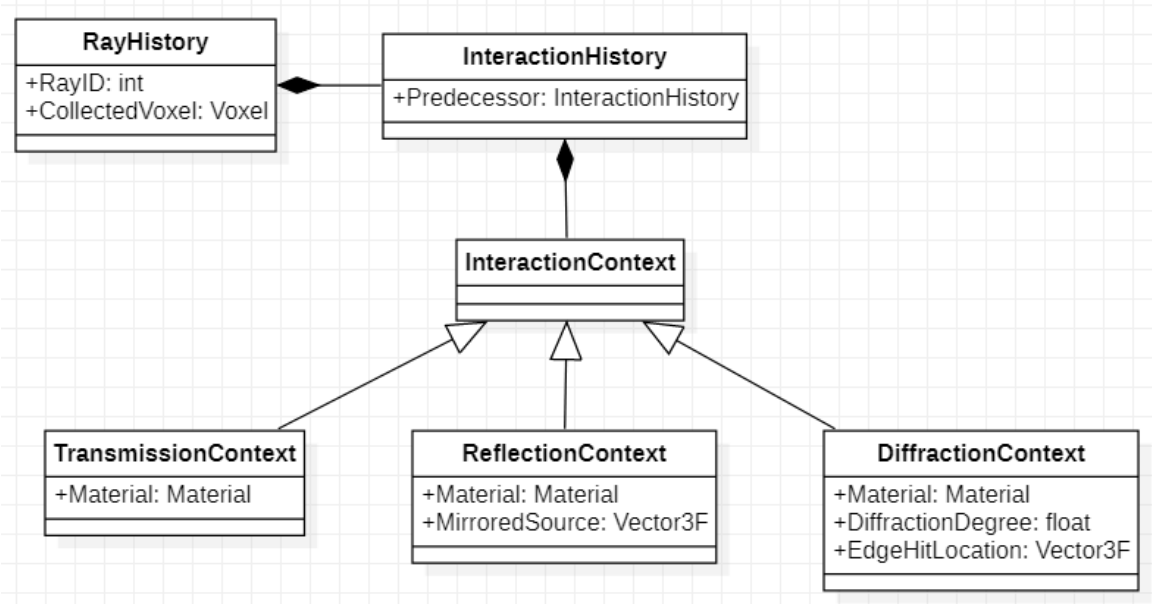


Figure 5-13 Ray history model

To collect the ray histories, the type `InteractionHistory` will be attached to `RayPayload`, so the chain of histories can be built up while new `RayPayloads` are generated from interactions. Then, the main pipeline given in Figure 5-12 will be reused, but this time the Ray Collector will be replaced with Ray History Collector, in which the ray history ended on each measurement point will be collected. However, the limitation of fixed-size buffer in the OpenCL kernel will cause a problem again, as the number of rays arriving at a measurement point is also variable. A solution is to make use of the concept of ray batch, as a ray batch is a collection of rays on the same path. Therefore, only one ray from a batch needs to be collected, and the buffer size can then be fixed. To maximize the performance, a few batches will be collected in parallel per kernel call, so the buffer size will be the number of batches multiplied by the number of measurement points. A simplified representation of the kernel is given in Code Snippet 5-3. The full kernel code is given in Appendix II TLARS OpenCL Kernels. In the inner loop of the kernel, if a ray enters a

measurement voxel, then the offset `bufferIndex` is calculated by the multiplication of the voxel index and ray batch index. In the end, the ray ID is inserted into the result buffer `CollectedIDsOnPoints` at the position `bufferIndex`.

```
function RayIDsOnPointsCollectingKernel(Grid, Ray,
MeasurementPoints, CollectedIDsOnPoints)
    for each Voxel in the Grid the Ray walks into
        for each Point in the MeasurementPoints
            if Point equals Voxel
                bufferIndex = Voxel.Index * Ray.BatchIndex
                Assign Ray.ID to CollectedIDsOnPoints at bufferIndex
```

Code Snippet 5-3 Ray history collecting kernel

5.5.4 Ray History Replay and GA Optimization

Once all the ray histories arrived on the measurement points are captured, the last step of the calibration is to replay the ray histories in the GA optimization. The problem domain of the optimization can be regarded as the full set of collected ray histories, and the solution domain is the material repository. The evaluation equation, which is the RMSE between the calculated pathloss values and the measured values, will be formed using the ray histories with the materials as the input.

Then each generation in the GA will create a new set of materials based on the mechanism introduced in Section 3.2.4. Upper and lower bounds will be given for the calibration, and 3000 iterations are used in the GA. The choice is based on the assumption of 10 unique materials, and the result from a few experiments, that the GA generations start to converge after about 2000 iterations.

In total, the full calibration process with the 3 steps introduced above is expected to have a complexity of about 3 times the one for a standard pathloss calculation in the

same scenario, due to that computational burden of each step is similar to perform a single pathloss calculation. Therefore, the difficulties discussed in Section 5.5.1 are efficiently addressed.

5.6 TLARS Results and Measurement Validation

In this section, the completed TLARS model will be used in different scenarios, to firstly test its performance, capability and its solution to the angular dispersion. Then the results will be validated with measurements in these scenarios. A comparison with PDP measured in a lecture room will be given as well.

5.6.1 Paris Scenario Calculation Result with Comparison to GORL and IRLA

Firstly, the calculation is repeated in the Paris scenario which was used in Section 4.9. The results at resolution 5m and 2m are shown in Figure 5-14 and Figure 5-15. To provide a fair comparison with GORL, the diffraction is turned off for TLARS. Comparing with the corresponding GORL results in Figure 4-13 and Figure 4-14, the ray separation caused by angular dispersion is largely eliminated by TLARS in both resolutions. This demonstrates the effectiveness of the proposed ray splitting method in solving angular dispersion.

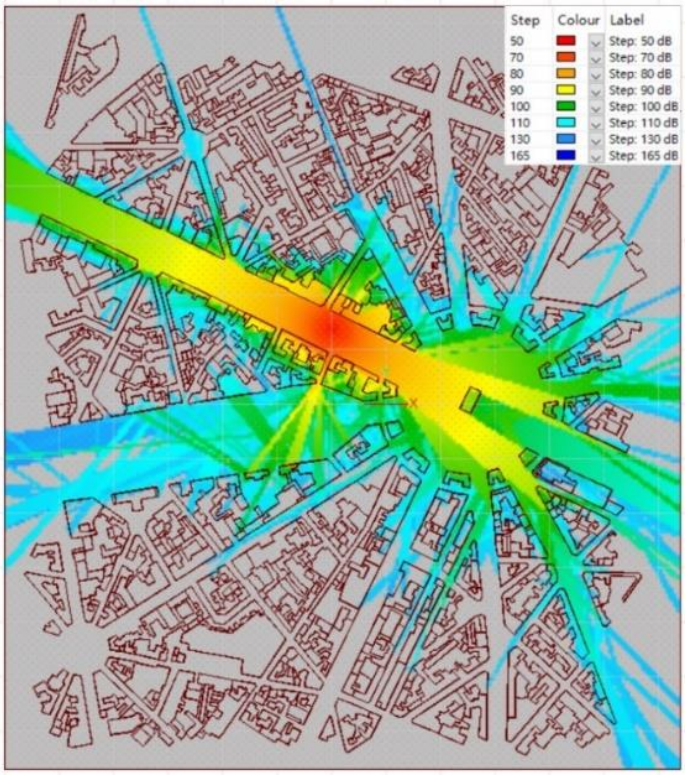


Figure 5-14 Paris TLARS result at 5m

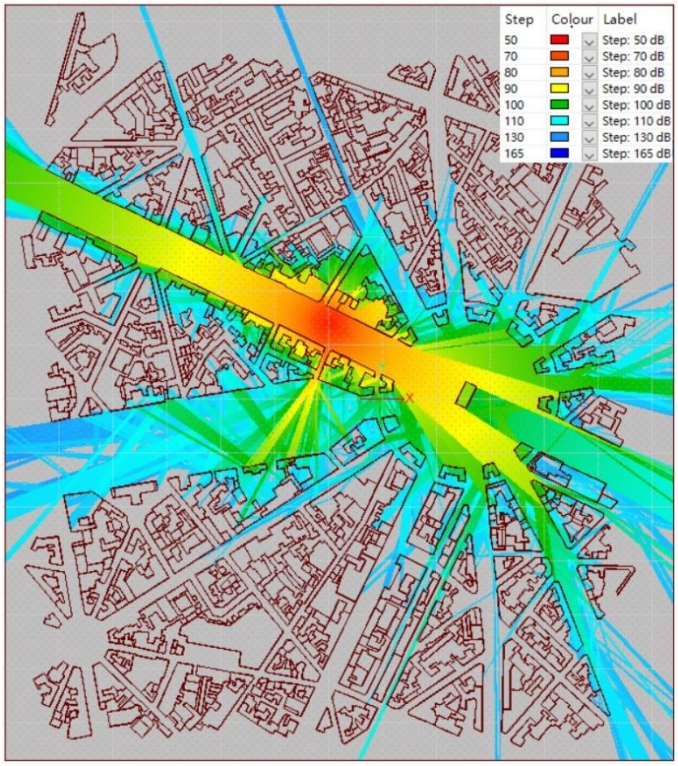


Figure 5-15 Paris TLARS result at 2m

Performance-wise, the comparison of time and memory consumptions for TLARS, GORL and IRLA is given in Table 5-1. Firstly, comparing to the original GORL, the speed of TLARS is slightly faster while the memory consumption is less than half of the GORL. It should be noted that the GORL doesn't solve the angular dispersion, and the equivalent solutions are the TLARS and the over launching method. Thus, it can be recognized that the TLARS provides a much more efficient solution with 2.73sec and 226MB at 5m resolution, comparing to 90.19sec and 3492MB used by over launching, not to mention that over launching is not able to be used at 2m resolution. Such significant improvement is coming from the reduced complexity provided by the ray splitting method, and the improved design that maximizes the parallelization with buffers to decouple the modules as discussed in Section 5.4.2.

	Time spent (sec)	Memory usage (MB)
TLARS – 5m	2.73	226
TLARS – 2m	8.68	478
GORL OpenCL (CPU) – 5m	3.29	429
GORL OpenCL (CPU) – 2m	10.01	1198
GORL OpenCL (CPU) Over launching – 5m	90.19	3492
GORL OpenCL (CPU) Over launching – 2m	NA	NA
IRLA – 5m	6.03	75
IRLA – 2m	84.1	486

Table 5-1 TLARS, GORL and IRLA compared in Paris scenario at 5m and 2m

Then comparing to IRLA, the TLARS is faster at both resolutions especially at 2m where the TLARS is about 10 times faster. On the other hand, from 5m to 2m resolution, the scale of increase in consumption is much larger for IRLA than for TLARS. This is due to that, as a vector model the complexity of the environment doesn't change with the resolution, while for a raster model the complexity of the environment is directly linked to the resolution. Thus, IRLA will suffer from a larger impact from the change in resolution.

After all, it is shown that TLARS as a full 3D vector model is performing much faster than IRLA as a 2.5D raster model.

A set of measured pathloss values from a transmitter at 1.8GHz is supplied for the Paris scenario [153]. The antenna is placed at the same location as in Figure 5-15 and is 7m above the ground. Comparing to the measurement, the TLARS result has achieved a ME of 5.67dB and an RMSE of 7.71dB before the calibration. Then after calibration, the ME is reduced to 4.40dB and the RMSE is reduced to 5.54dB. The plot of the pathloss values before and after calibration against the measurement points is given in Figure 5-16, and the histogram of the error between the calibration and measurement is given in Figure 5-17.

From the calculated errors, good accuracy is achieved by TLARS even before the calibration. This is the advantage of a deterministic model which is less sensitive to environmental change than the empirical models. With calibration, a much closer fit to the measured values is achieved, which can also be seen from the plot. Meanwhile, the calibration process takes 7.9sec, which is about 3 times the pathloss calculation time, as expected in Section 5.5. Due to the 3 stages introduced in the calibration process, the reduction of the scope for tracked rays does improve the calibration efficiency.

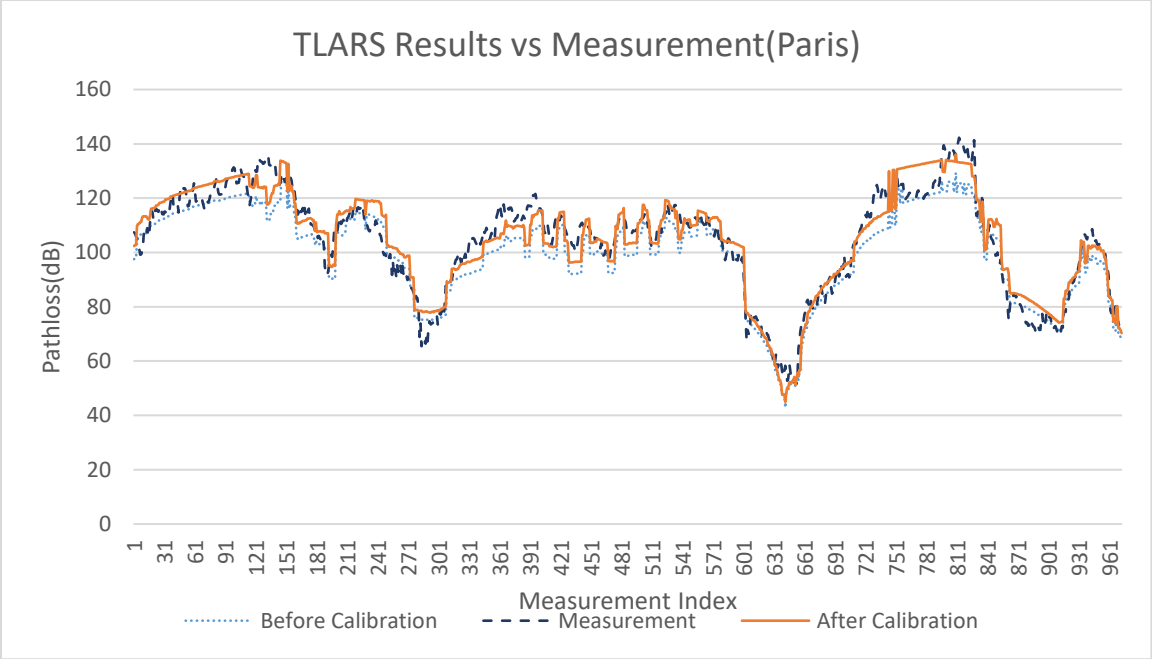


Figure 5-16 Paris measurement vs. TLARS results before and after calibration

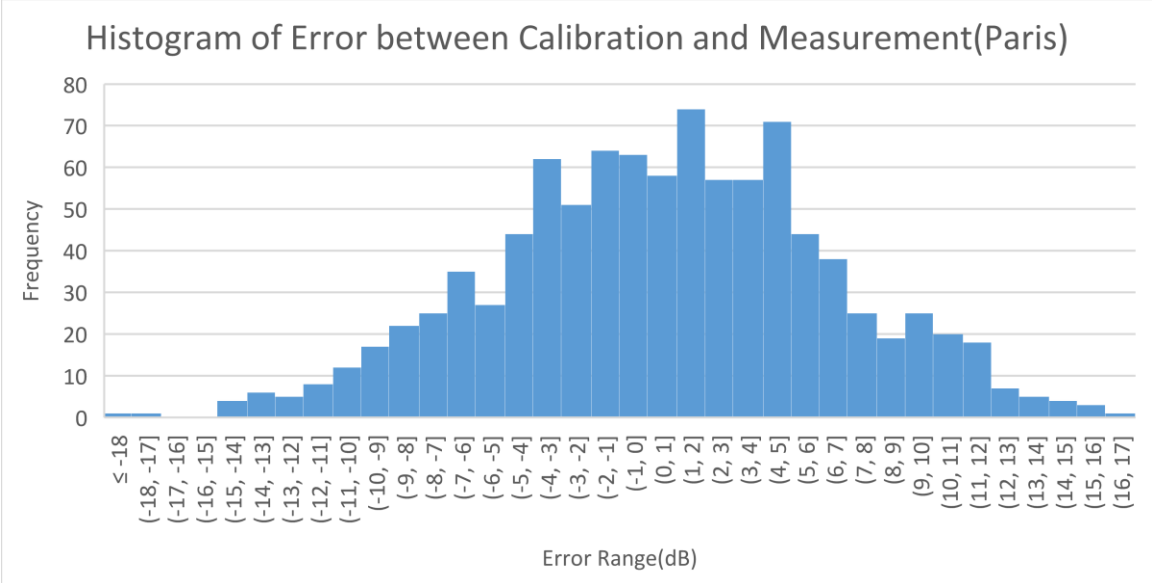


Figure 5-17 Histogram of error between the calibration and measurement in Paris

5.6.2 Indoor Office Scenario Result with Comparison to IRLA

In this section, the result of TLARS is compared with IRLA again in an indoor office scenario together with measurement data. As discussed in Section 3.2.2, part of the

IRLA speed performance is benefited from its 2.5D simplification, that a 3D ray would be converted to a 2D ray in the horizontal plane after any reflections. Such conversion should introduce less impact in the indoor environment as a floor is bounded by the ceiling and ground. Therefore, it is worth to check the accuracy of the results from TLARS and IRLA in the indoor environment, where the IRLA is well established with. The modelled indoor office has a dimension of $38m \times 42m \times 4m$, as shown in Figure 5-18. Two transmitters are placed at the top-left and bottom-right of the corridor and are under the ceiling at a height of $3.8m$.

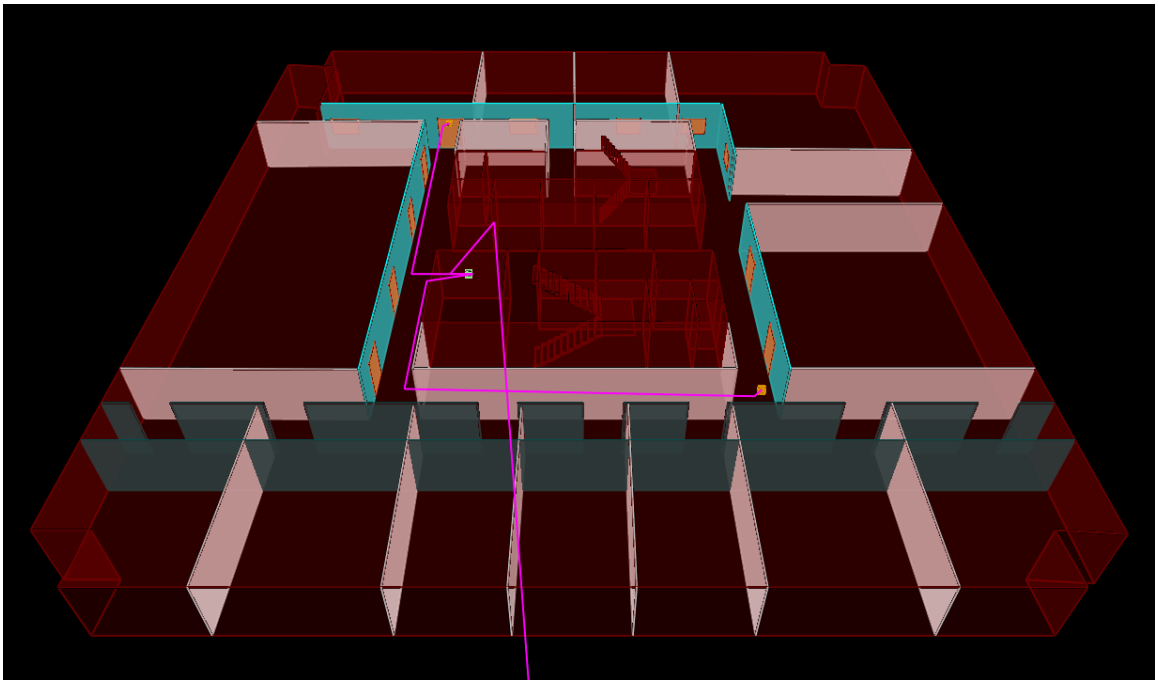


Figure 5-18 Indoor office scenario

The calculated results from TLARS and IRLA after calibration are shown at the top of Figure 5-19, where at the bottom is the measurement data. 19 measurement points are collected in different rooms throughout the floor at a height of $1.7m$ above the ground. To validate the accuracy of the calculated results, their differences from the measured values

are plotted in Figure 5-20, where the TLARS result presents a better approximation to the measurement. The ME and RMSE for TLARS and IRLA are listed in Table 5-2, which again shows that TLARS has higher accuracy than IRLA in this scenario. This could be a result of the advantage of a vector 3D model over a raster 2.5D model.

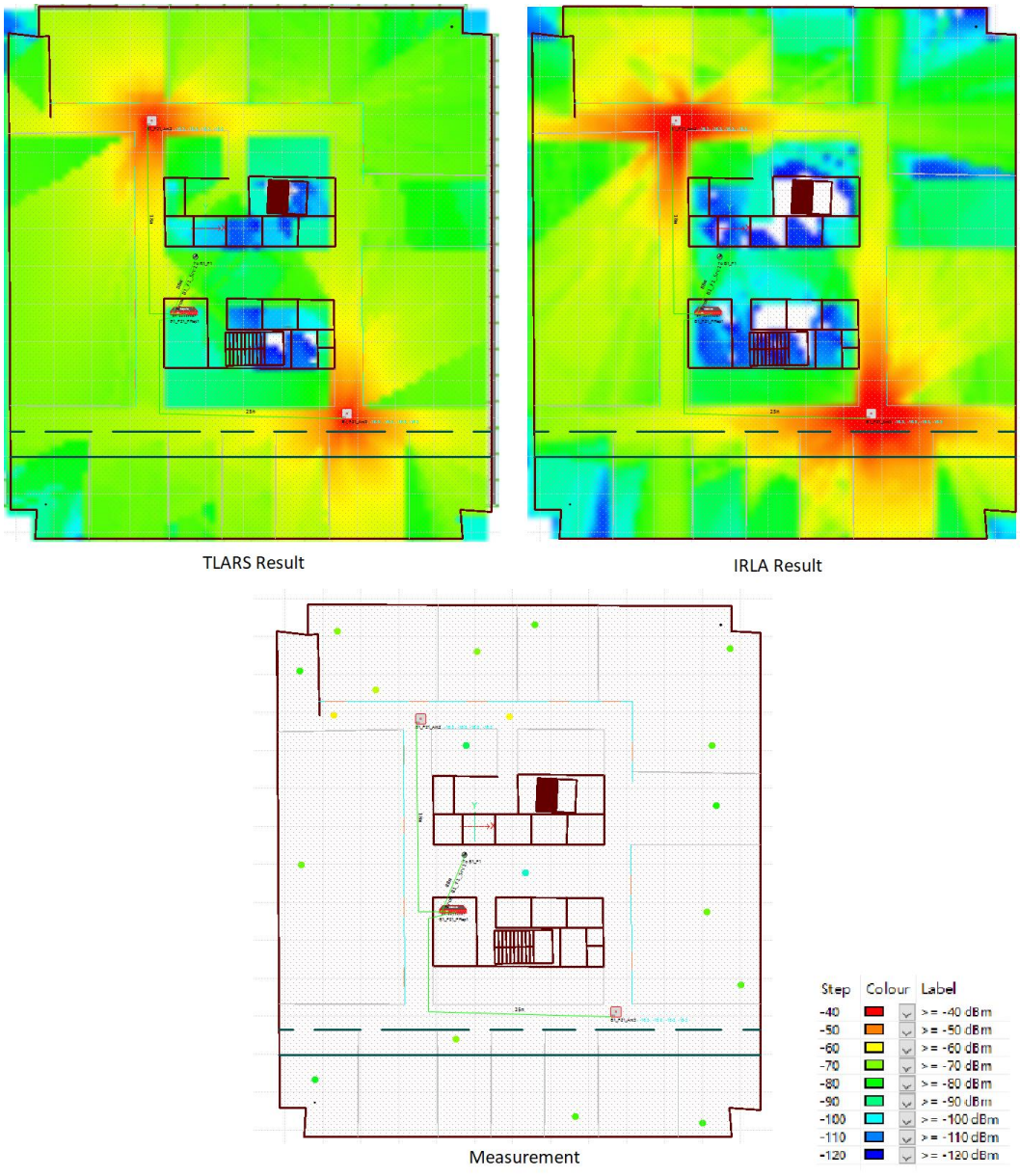


Figure 5-19 Office scenario calculation results at 0.5m resolution from TLARS and IRLA compared with measurement

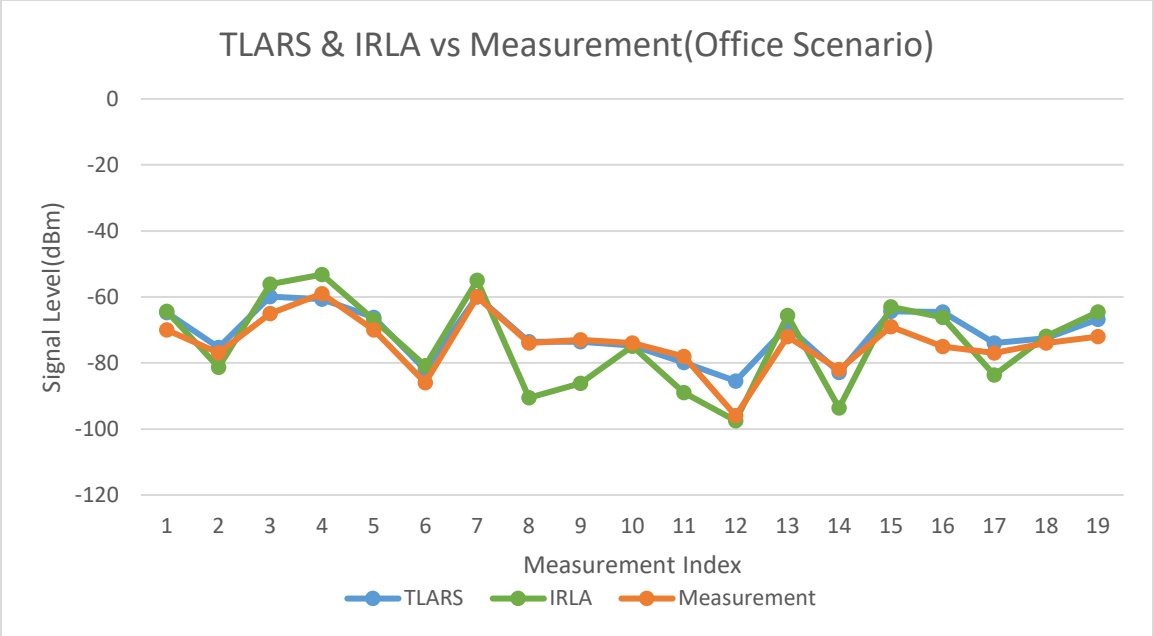


Figure 5-20 Office measurement vs TLARS and IRLA results after calibration

	ME	RMSE
IRLA before calibration	8.75dB	10.09dB
TLARS before calibration	8.11dB	9.32dB
IRLA after calibration	6.84dB	7.91dB
TLARS after calibration	3.39dB	4.49dB

Table 5-2 Errors between calculated results and measurement in the office scenario

5.6.3 Munich Scenario Result

The Munich scenario provided by COST 231 [41] is a widely cited scenario with a set of measurement campaigns provided. It has a dimension of $2400m \times 3400m \times 100m$, with 21619 building polygons. The transmitter is placed at 13m high operating at 947MHz. The TLARS calculation result at 5m resolution is shown in Figure 5-21, with comparison to the measurement route. The calculation takes 27sec and 1336MB, which are higher than the Paris scenario, due to the increased scenario dimension.

Meanwhile, the plot of the measured values against calculated results before and after calibration is given in Figure 5-22, and the histogram of the error between the calibration and measurement is given in Figure 5-23. A good match is shown both in the heatmap comparison and value comparison. Before calibration, the ME is 7.15dB and the RMSE is 9.36dB. After calibration, the ME is reduced to 4.79dB and the RMSE is reduced to 5.88dB. Therefore, the comparison proves the accuracy of TLARS in medium-range outdoor scenario.

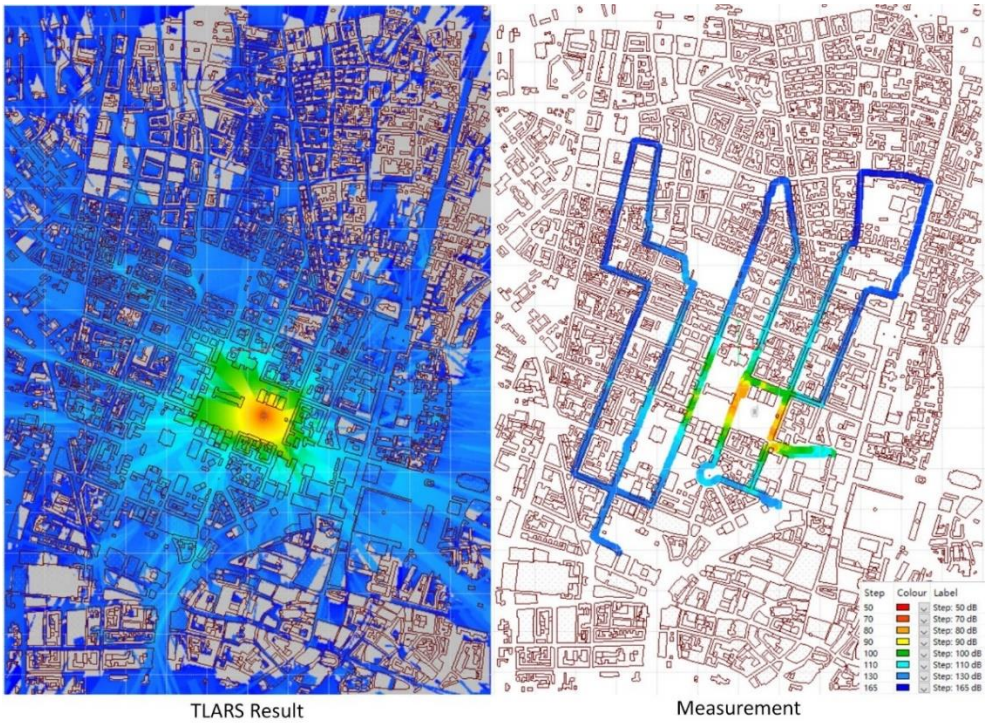


Figure 5-21 Munich calculation result at 5m compared with measurement

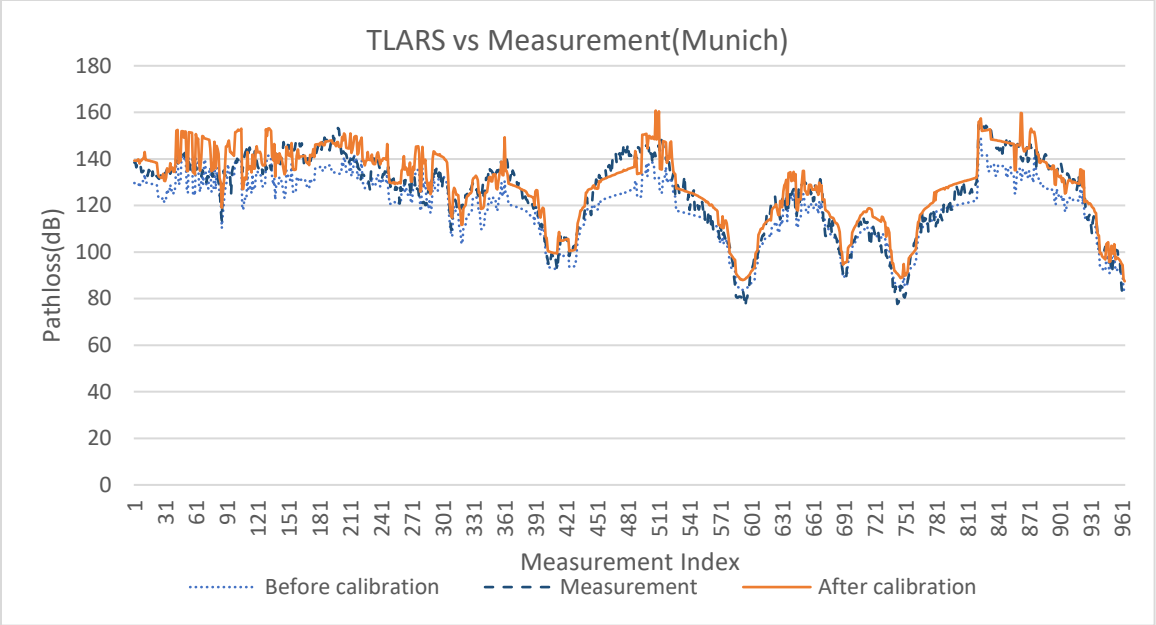


Figure 5-22 Munich measurement vs. TLARS results before and after calibration

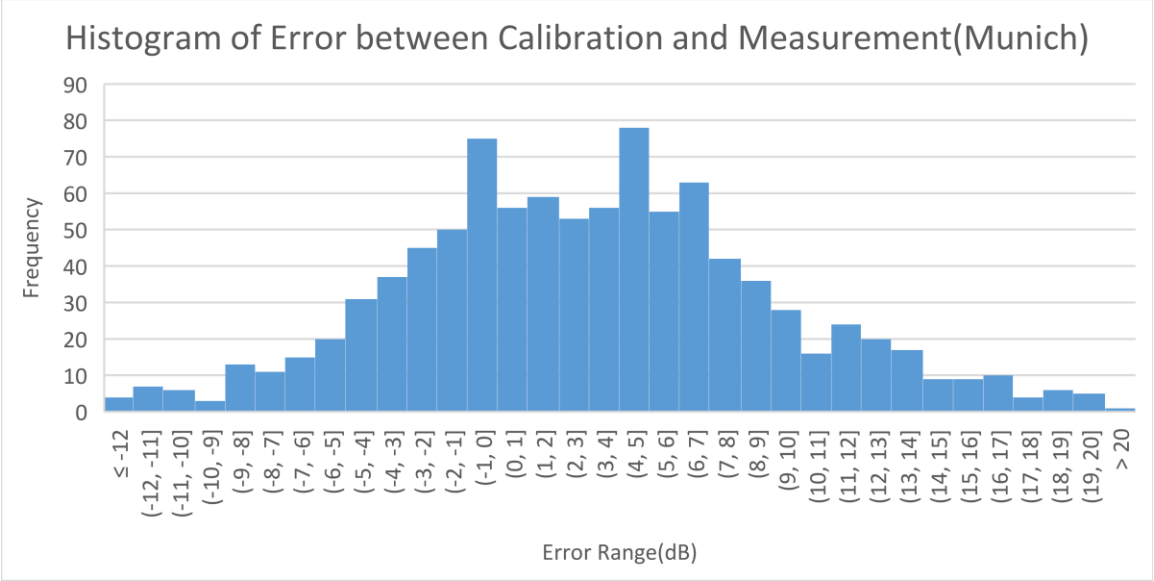


Figure 5-23 Histogram of error between the calibration and measurement in Munich

5.6.4 Santiago Scenario Result with Consideration of Terrain

So far, the two used outdoor scenarios are not supplied with terrain data, which is acceptable if the terrain is flat in the original environment. In this section, the Santiago

scenario will be used with the consideration of terrain. The modelled environment has a dimension of $3260m \times 3060m \times 400m$, including the variation in height from the terrain. The polygons included in the scenario are composed by 223,831 building polygons and 5,124,733 terrain polygons. The number of terrain polygons is about 23 times the building polygons and would add excessive complexity to the scenario.

The calculated pathloss result at 5m resolution is shown in Figure 5-24, where the transmitter is 30m above the ground on the top of a building and operates at 2.6GHz. The calculation takes 276sec and 3926MB. Such significantly increased consumption is coming from the big scenario dimension, especially the 400m variation in height, and much denser polygons, i.e. the Santiago scenario has about 247 times the polygons in Munich scenario. Also, the modelling of terrain by uniform triangles is not efficient, which would be critical for high-resolution calculation. Therefore, it should be improved with smooth terrain modelling, e.g. the adjacent uniform triangles on the same plane should be merged into non-uniform triangles.

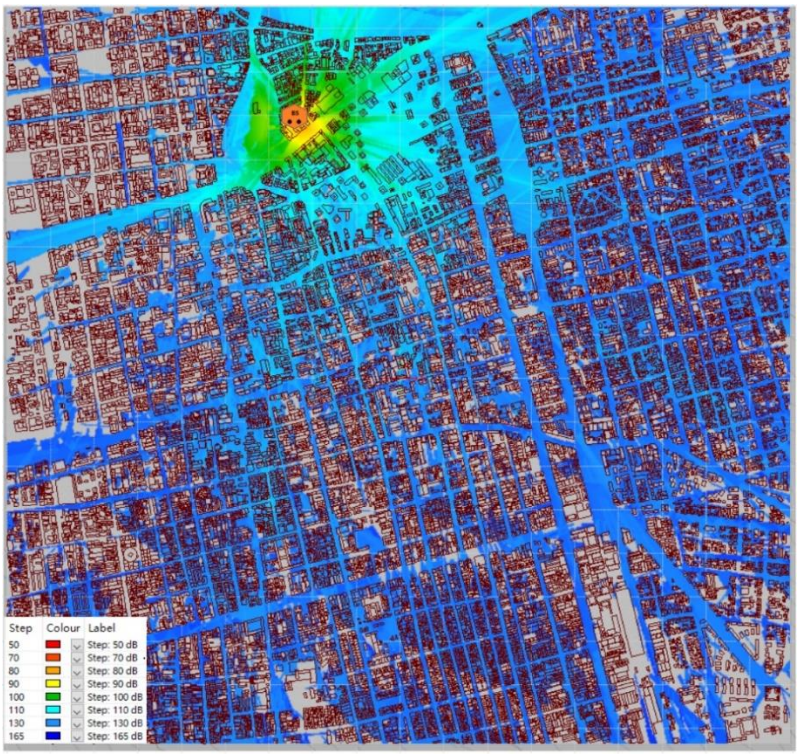


Figure 5-24 Santiago pathloss result at 5m resolution

A measurement from a driving test is provided for the scenario, as shown in Figure 5-25. Comparing to Figure 5-24, the calculated result is showing a good match with the measurement. Then the plot of the calculated pathloss against the measured values on each point is given in Figure 5-26. In overall, before calibration the RMSE is 9.63dB and the ME is 7.78 dB, while after calibration the RMSE is 7.08dB and the ME is 5.60dB. The errors are larger than the previous two outdoor scenarios but are still maintained within a good range.

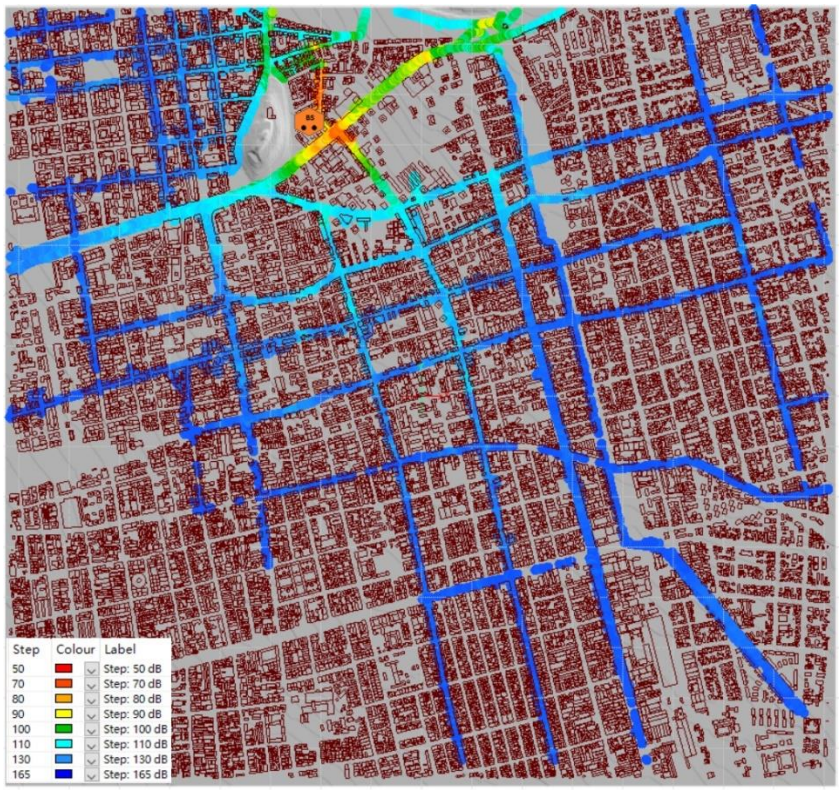


Figure 5-25 Santiago measurement points

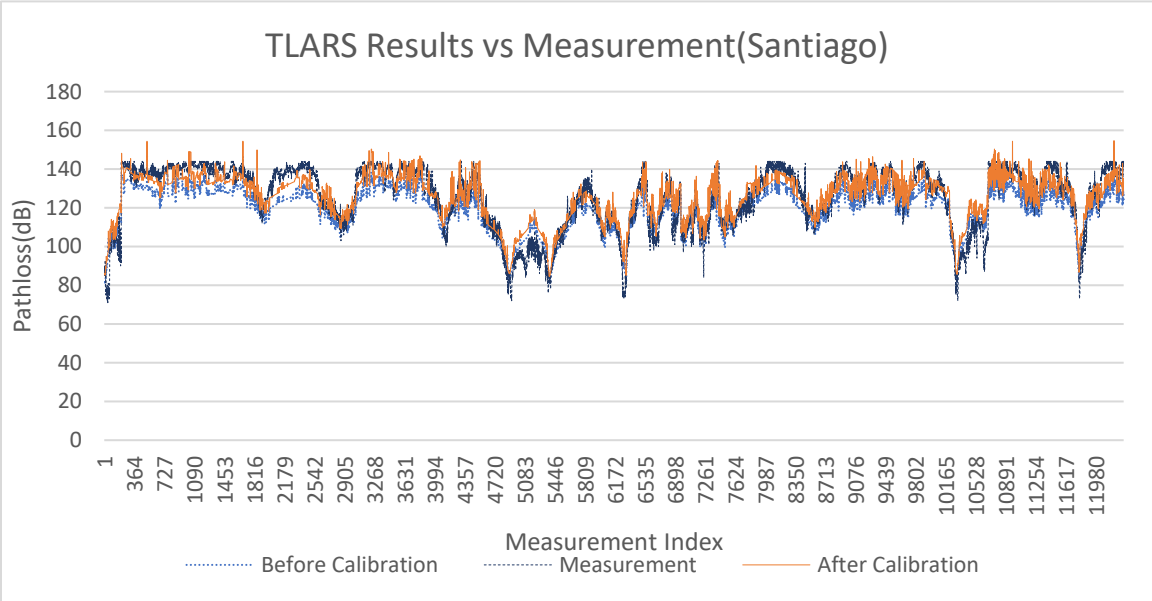


Figure 5-26 Santiago measurement vs. TLARS results before and after calibration

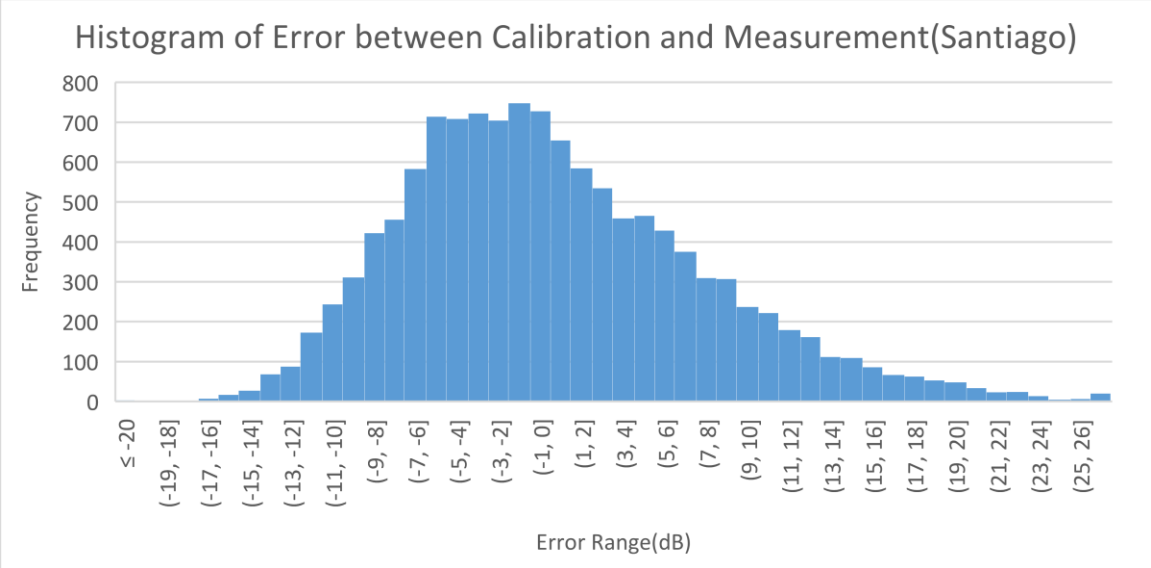


Figure 5-27 Histogram of error between the calibration and measurement in Santiago

5.6.5 Multipath Result Compared with Indoor PDP Measurement

In the end, an indoor scenario is used to validate the multipath calculation of TLARS. The floor plan is shown in Figure 5-28, which is a lecture room with a dimension of $8.6m \times 7m \times 2.8m$. The green dots represent the locations of the measurement and the red dot represents the receiver. An omni-direction antenna operated at 39GHz is placed at the measurement points at 1.6m high, and a directional antenna at 1.7m high is used as the receiver.

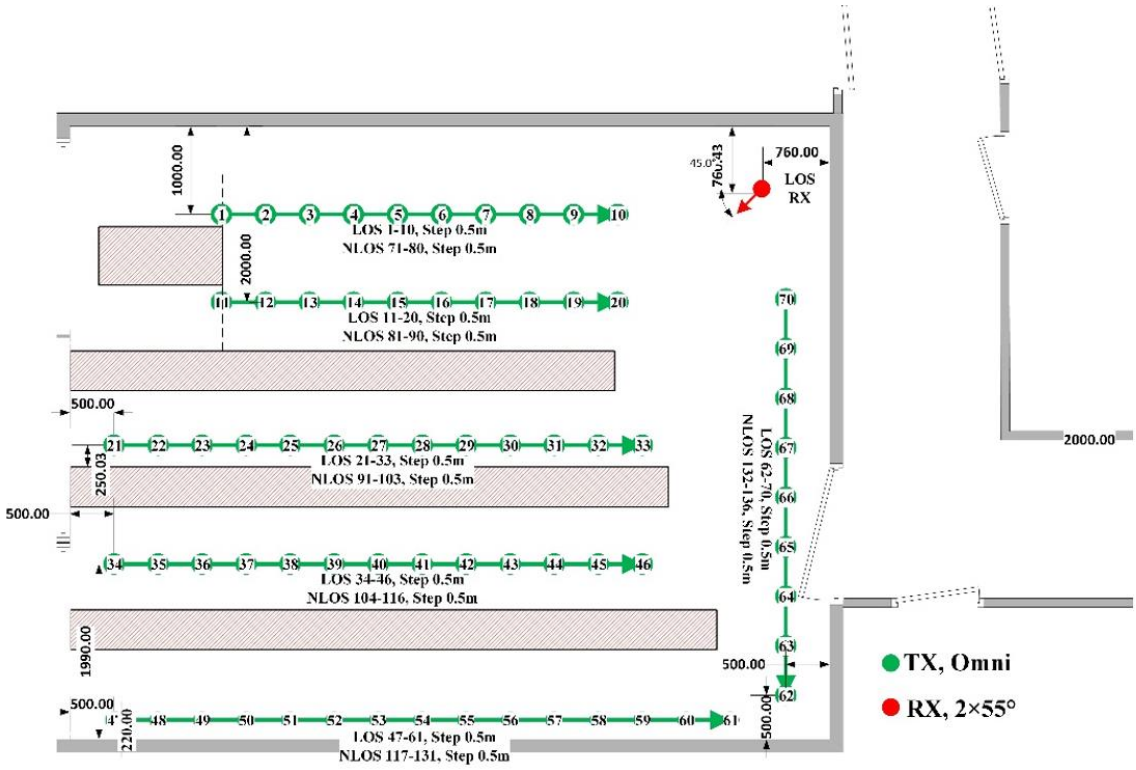


Figure 5-28 The floor plan for PDP measurement

Firstly, the PDPs from measurement and TLARS calculation are compared at 4 points. For the ease of comparison, the PDP time delays are aligned by the first peak from the measurement and calculation, so the plotted time delay is relative. As shown in Figure 5-29, where the blue line represents the measurement and the asterisks represent the calculation, a good match is achieved between them. The first few major peaks in the measurement are well approximated by the asterisks both in time and in power strength. There is no calculation result for the latter half of the measurement, i.e. beyond 100ns delay, due to that the maximum number of reflections is limited to 5.

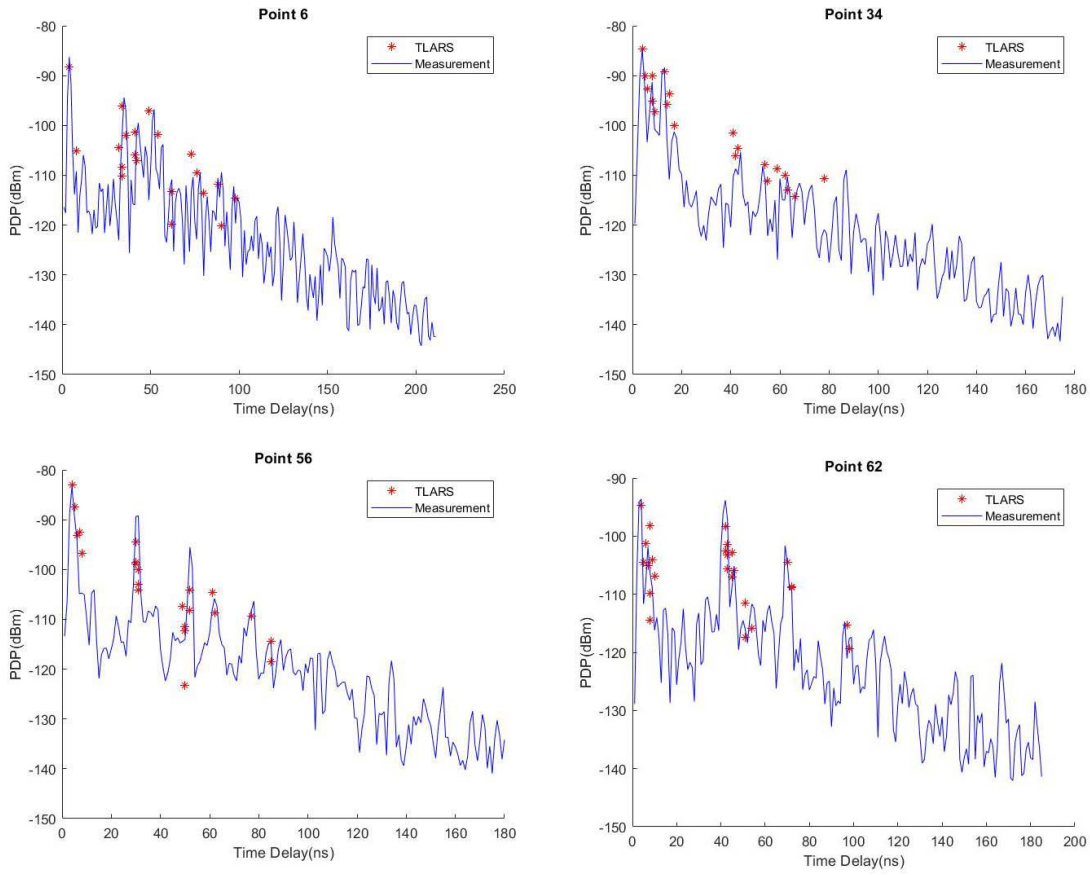


Figure 5-29 PDP comparison between TLARS and measurement

In overall, the RMS delay spread for the 70 points is calculated for the TLARS and measurement. The latter half the measurement is omitted to match the limited interactions considered in TLARS. The comparison is shown in Figure 5-30, where the circle denotes the TLARS and the diamond denotes the measurement. The calculated results do well match the measurement through all the points. Therefore, the comparisons above present the accuracy of TLARS in providing multipath information in an indoor environment.

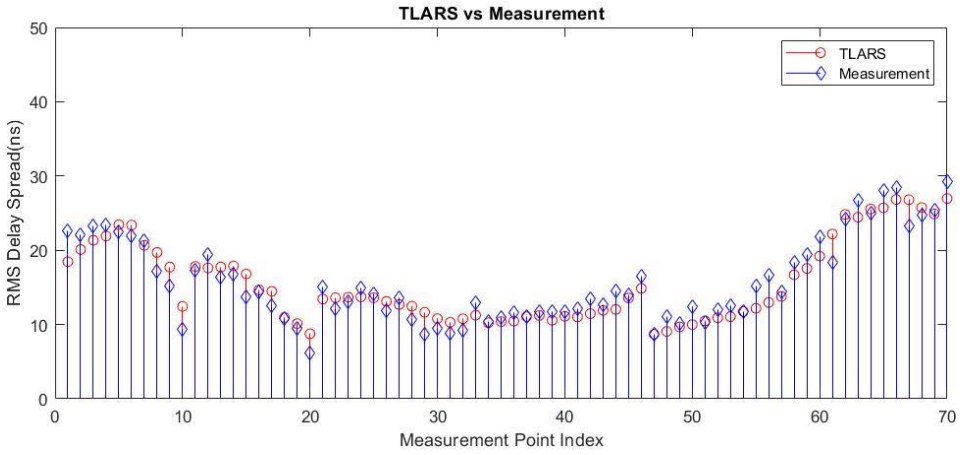


Figure 5-30 RMS delay spread comparison between TLARS and measurement

5.7 TLARS Limitations

Although TLARS is shown to have good accuracy and fast speed in Section 5.6, its mechanism is still subjected to limitations under some circumstances. Intrinsically, the performance of TLARS is benefiting from the adaptive splitting process, that four child rays are fully represented by their parental ray during the preceding trajectory. As a result, the blockage of a parental ray will result in the blockage of its four child rays although some of the child rays may be free from the blockage if the actual trajectory is drawn from the tube source to the child ray. Therefore, the TLARS mechanism could be very efficient during the free space propagation but may subject to some limitations on resolving the surface of a blockage. This section will present those limitations anticipated and found during the usage of TLARS.

5.7.1 Ripple effect from incidence on terrain slope

When calculating the signal on the surface of a terrain slope, the TLARS may treat part of the slope as an NLOS area, although the LOS path is presented. The cause of such error can be explained in Figure 5-31. Firstly, an initial ray is emitted from the signal source and split into two child rays in the 2D view. Then following the TLARS mechanism, an actual ray path shown as the green line is approximated by the top one of the child blue rays. Therefore, the blue child ray will predicate the middle section of the slope as NLOS while the green ray is LOS. When calculating a real scenario with terrain presented, such error will be shown as signal gaps with a ripple-like pattern along the terrain contour lines, as shown in Figure 5-32.

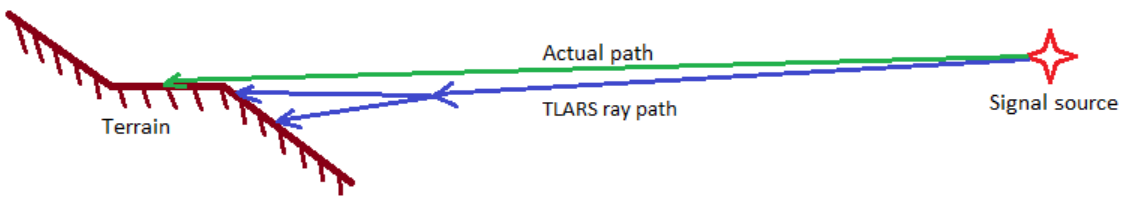


Figure 5-31 Blockage of a split ray while the actual path is clear

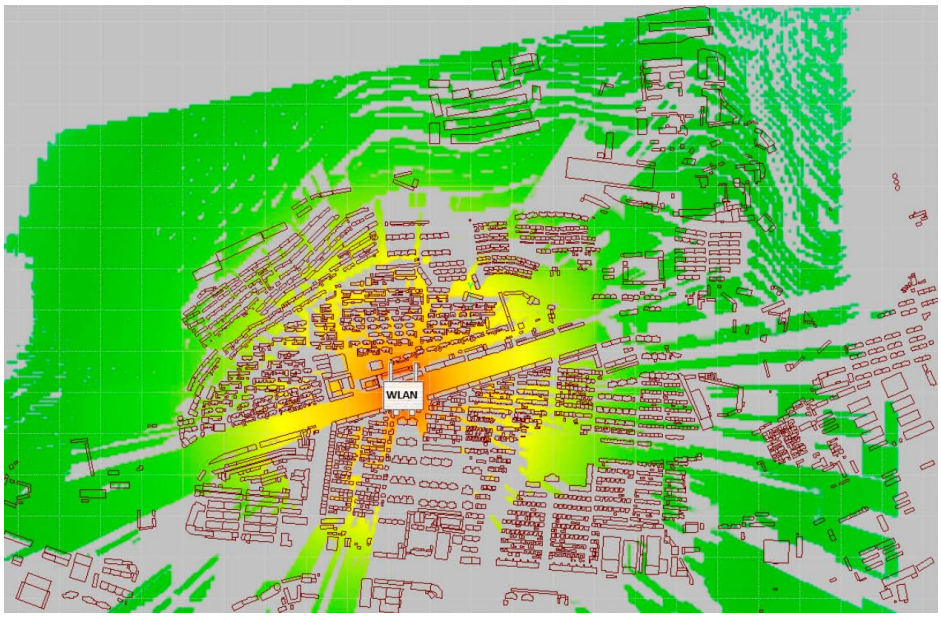


Figure 5-32 Ripple effect on terrain contour lines

One way to reduce the ripple error is to launch the split child rays back from the tube source. In the case of the initial rays, the tube source will be the signal source. Therefore, in the example shown in Figure 5-31, the top blue child ray will be replaced with the green ray, and the LOS section of the slope will be calculated correctly. This is exactly the second method of launching split rays in Figure 5-3. Thus, also as discussed in Section 5.1, significant calculation overhead could be introduced by such method if the tube source is a mirrored source after reflections. But, if the method is only applied to the initial rays, then the calculation would be free from the overhead issue. Figure 5-33 presents the application of the solution on initial rays, and it can be seen that the ripple effect is removed.

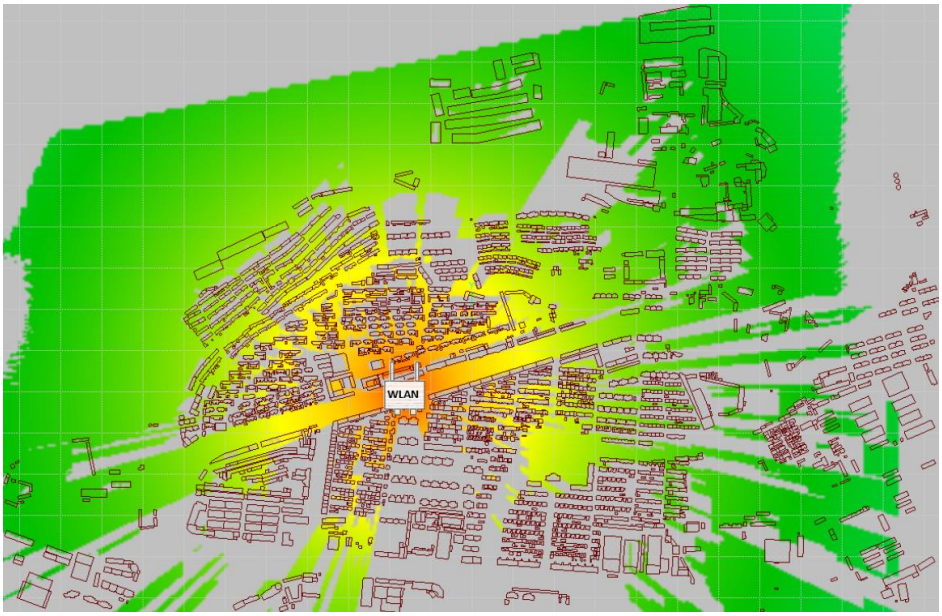


Figure 5-33 Ripple effect removed with split ray launched from signal source on terrain hit

5.7.2 Signal Gaps on Surface during Shallow Incidence

The splitting algorithm designed in TLARS guarantees that the distance between two adjacent rays will always be less than a given resolution. However, if the incident angle of a ray on the surface is very shallow, then the hit points of two adjacent rays could result in a distance much larger than the resolution, as shown in Figure 5-34. Then in Figure 5-35, the impact of the gap after a shallow incidence is shown in a calculated result. In this figure, the used scene is a simple rectangle room, and the transmission, reflection and diffraction are disabled for a clear presentation of the impact. It can be seen that signal gaps are presented on the top wall on the left and right sides, where the incident angle of the ray is becoming large.

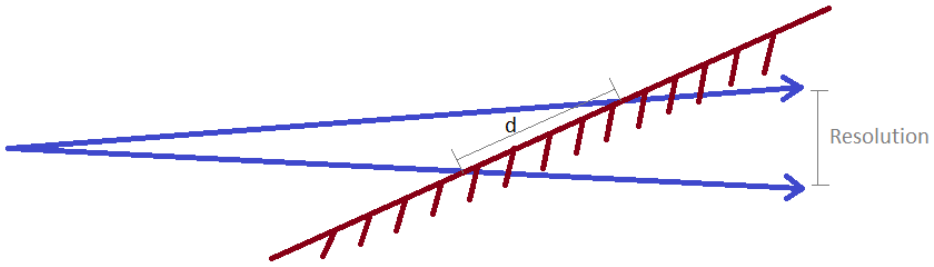


Figure 5-34 Gap larger than resolution during a shallow incidence

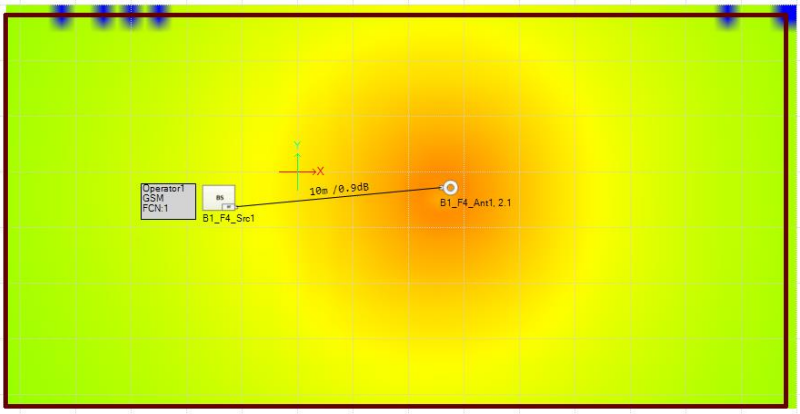


Figure 5-35 Signal gap on the wall surface in the horizontal view

To overcome such an effect, one possible solution is to launch interpolated rays within the tube on a detected shallow incidence. The shallow incidence can be found from calculating the d denoted in Figure 5-34, by knowing the incident angle, the hit position and the current tube end. However, such interpolation will inevitably incur a lot of performance overhead as the number of rays can be a few times of the original algorithm for shallow incidence. The study and implementation of the solution will be a future work of this research. For the existing implemented TLARS, the occurrence is only narrowed to the pixels next to the wall, and the impact will be eased if the reflection and transmission are calculated.

5.8 Conclusion

In this capture, the innovative tube launching model TLARS is proposed. The tube-based method provides continuous modelling of the EM wave propagation in the 3D volume. Then as approached by adaptive ray splitting, the model can be implemented with efficient ray-based mechanisms including spatial indexing, ray triangle intersection and grid traversal as introduced in Chapter 4. Furthermore, with the improved module design maximizing the parallelization, the TLARS is presented with very fast speed in the simulations.

The simulation metrics are summarized in Table 5-3. A good match with the measurement results is achieved both before and after the calibration in different scenarios. Moreover, in an indoor scenario, the calculated multipath results are shown well aligned with the measurement PDPs. Thus, the accuracy and versatility of TLARS are presented. However, the limitations especially the gaps caused by shallow incidence should be studied in the future for a feasible solution. Also, the speed drop in the Santiago scenario is indicating that the current modelling of the terrain data is not efficient, especially when a high calculation resolution is used. An improved terrain modelling mechanism should be the future work of this study as well. After all, the TLARS is shown to be capable of solving the problem identified in this study, which is to provide efficient full 3D radio prediction.

	Time spent	Memory consumed	RMSE (before calibration)	RMSE (after calibration
Paris scenario (5m)	2.73sec	226MB	7.71dB	5.54dB
Munich scenario (5m)	27sec	1336MB	9.36dB	5.88dB
Santiago scenario(5m)	276sec	3926MB	9.63dB	7.08dB

Table 5-3 Summary of TLARS simulations in different scenarios

6 Outdoor-to-Indoor and Indoor-to-Outdoor Radio Prediction Based on Multi-Resolution Tube Launching

For the medium-range radio prediction studied in this thesis, one typical use case is the transition between indoor and outdoor scenarios, as the outdoor cell coverage in indoor and indoor small cell leakage in the outdoor matter in the HetNet deployment. However, such transition encounters a problem of different resolutions. As high-resolution environmental information would be available for specific buildings, in which high-resolution radio prediction are also expected, while vice versa for outdoor scenarios. If such a hybrid scenario is fully calculated aligned with the high resolution required in indoor, the complexity would become overwhelming for the associated outdoor area, according to the equation (4-16). The limits of existing studies [95-97] are discussed in Section 2.2.3. Due to the introduction of hybrid methods, the existing models add artificial transition points on the boundary between the indoor and outdoor scenarios. This will inevitably distort the ray direction in a deterministic approach.

Benefiting from the ray splitting method proposed in Chapter 5, the tube model in TLARS can fit different resolutions adaptively by cutting a tube to fit a smaller resolution or extending a tube to fit a larger resolution. Therefore, this study will propose the multi-resolution tube launching model (MR-TLARS) providing a multi-resolution method to perform the radio prediction with seamless transition between different resolutions in a hybrid scenario.

6.1 Outdoor-to-Indoor Radio Prediction with Refined Tube Launching

The hybrid environment is usually composed of a large-scale outdoor scenario and a few indoor scenarios. In the outdoor scenario, the buildings are given with their bounds, among which a few would have their associated indoor scenarios. To transit the calculation from outdoor to indoor, the key method in MR-TLARS is to generate the indoor refined high-resolution tubes during outdoor calculation with low-resolution tubes. The process can be divided into four steps: refined tubes generation triggering, refined tubes casting, refined tubes capturing and refined tubes re-launching in indoor.

6.1.1 Refined Tube Generation Triggering

The tube model provided in TLARS is self-contained to resolve environment entities and provides pathloss values within its volume by the given resolution. Also, the actual visibility condition, i.e. LOS and NLOS, is persisted in the tube as the source of the tube is always kept and updated after reflection and diffraction. Therefore, the refined indoor tubes can be re-generated from the source within the cross-section of a corresponding outdoor tube. Assuming $R_{outdoor}$ as the outdoor resolution and R_{indoor} as the indoor resolution, then the number of indoor tubes that would be generated from one outdoor tube is $(\frac{R_{outdoor}}{R_{indoor}})^2$. The method could result in 100 indoor tubes if the outdoor and indoor resolutions are 5m and 0.5m. Such complexity is excessive and aligned with the Equation (4-16). Therefore, the scope of refined tubes generation needs to be limited.

Thus, MR-TLARS will only generate the refined tubes that have a chance of hitting the outdoor bound of a corresponding indoor building. As given in Figure 6-1, the outdoor polygons that represent the bound of a corresponding indoor will be marked with a flag. Then outdoor tubes hitting the marked polygons will trigger the refined tubes generation,

while the ones hitting the normal outdoor polygons won't trigger. This is a reasonable approximation, as for those not triggered tubes, if their child tubes after certain interactions end up with hitting the marked polygon, they will still trigger the generation by that time. In this case, the refined tubes are reduced from irrelevant generation and early generation.

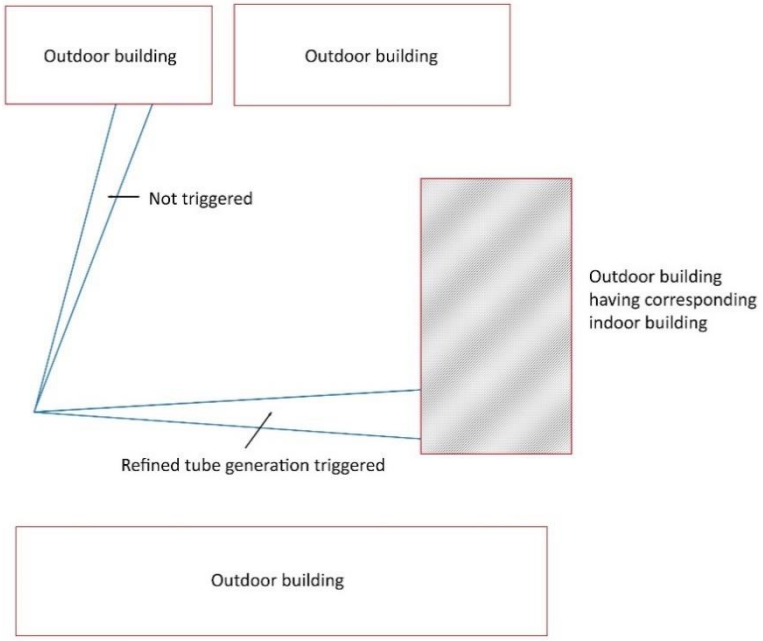


Figure 6-1 Refined tube generation triggered on the outdoor bound of corresponding indoor building

Figure 6-2 Outdoor tube around the marked polygon misses the triggering

However, it is found that the triggering on marked polygons may miss candidate tubes around the edge. Because a tube can only be captured if its representing ray hits the marked polygons. As shown in Figure 6-2, the missed tube could have overlap with the building bound. Thus, no refined tubes will be generated for the overlap area and lead to signal gaps in the indoor prediction result.

To solve the issue of missed triggering, the triggering condition is changed from hitting on marked polygons to intersecting an expanded bound. The original AABB of the surrounding polygons for a marked building is expanded by one $R_{outdoor}$ to form the expanded bound. Therefore, the partially overlapping tubes won't be missed from triggering. But such a triggering condition would introduce additional computational burden to the model. One solution is to convert the bound into scenario polygons to perform the check as a ray-triangle intersection in the kernel. But this is not ideal as the bound is not a real scenario entity, the termination of ray would be disruptive. Another option is to perform the ray-box intersection test side-by-side with the main pipeline on the CPU. But such a test does not need to be performed on all the rays. To improve the efficiency, the rays in a batch will be tested only if any one of the rays does hit a marked polygon. Thus, no additional operation is introduced apart from the necessary ray-box test.

6.1.2 Refined Tube Casting

Once an outdoor tube meets the triggering condition, the refined tubes need to be generated within its volume. The generation method is very similar to the original tube splitting method in TLARS. As shown in Figure 6-3, instead of splitting into 4 child tubes, the parent tube is split into $\frac{R_{outdoor}}{R_{indoor}}$ tubes on each dimension on its cross-section, resulting

in $(\frac{R_{outdoor}}{R_{indoor}})^2$ refined tubes in total. As for the rays representing the refined tubes, they would start from the source location of the outdoor tube, instead of the end of the parent ray. This is due to that the purpose of the refined tubes is to repeat and resolve the parent tube with a finer resolution, instead of propagating forward in the TLARS.

This multi-resolution generation method is very similar to the second ray splitting option given in Figure 5-3. As discussed in Section 5.1, such option will encounter a problem after reflection, that the reflection of the parent ray cannot imply the reflection of all the child rays. Therefore, if the most recent interaction of a triggered outdoor tube is a reflection, the corresponding reflection polygon needs to be recovered. Then for the generated refined tubes, their representing rays need to be performed with an intersection test against the reflection polygon. In the end, only the intersected rays could be reflected and therefore cast.

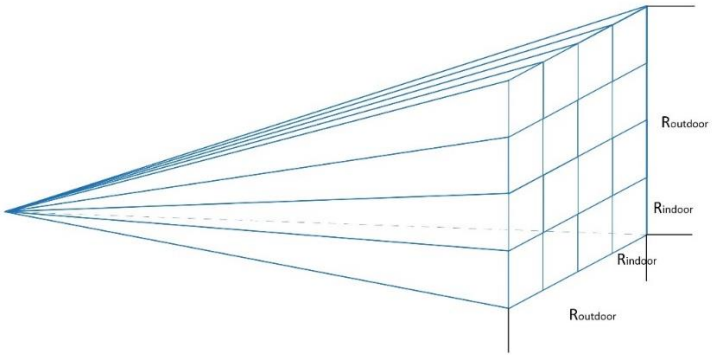


Figure 6-3 Refined tubes generation for one outdoor tube

However, the refined tubes generation is still inefficient if the source of the outdoor tube is very close to the building bound. In this case, the refined tubes would be so dense that they would never have a chance to split once in the indoor propagation. Such dense tubes will introduce overhead with unnecessarily high resolution for indoor calculation. An

example is given in Figure 6-4, for an outdoor tube source very close to the marked building, its designed split end would be far behind the targeted indoor building, and only until this end the generated refined tubes will need to split. Such inefficiency could be significant, as in the outdoor scenario it is very likely to have outdoor buildings around a marked building, and diffraction on an edge would create many tubes along the edge.

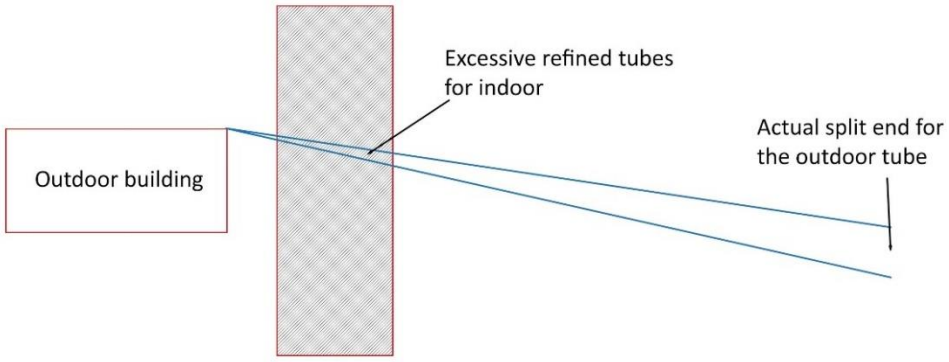


Figure 6-4 Excessive refined tubes generated from diffraction on an outdoor building close to the marked building

Therefore, the refined tubes generation method is improved with an adaptive approach. Once the generation is triggered, the refined tubes won't be generated targeting the cross-section at the end of the outdoor tube. Instead, an intermediate cross-section within the volume of the marked building will be targeted. Assume the length of the outdoor tube is t_0, and the length from the source to the intermediate cross-section is t_m, then the number of generated refined tubes is

$$Num_{refined} = (\frac{R_{outdoor}}{R_{indoor}})^2 \times (\frac{t_m}{t_0})^2 \quad (6\text{-}1)$$

Thus, for a nearby outdoor tube where the t_m would be small comparing to t_0, the number of refined tubes $Num_{refined}$ would be much less than the original $(\frac{R_{outdoor}}{R_{indoor}})^2$. It should be noted that t_m doesn't have to be calculated from the exact middle of the outdoor tube during the indoor building, as the function of t_m is to provide a guide. So, the length from a source to the centre of the marked building is used to approximate the t_m.

6.1.3 Refined Tubes Capturing

Once an outdoor tube generates the refined tubes, they won't all hit on the surface of the marked building, even the parent outdoor tube does hit. Two such examples are given in Figure 6-5. For the top tube, the outdoor ray hits the marked polygon and refined tubes are generated. However, the refined rays within the outdoor tube could be partially blocked by other outdoor building. This is because a low-resolution outdoor tube is now being resolved by high-resolution indoor tubes. As a benefit, the impact from outdoor entities can be calculated with higher resolution for marked indoor buildings. For the bottom tube in the figure, the refined tubes generation is triggered by the parent ray being intersected with the extended bounding box. Therefore, not all the refined tubes will hit the actual marked building, and those tubes missing the indoor building won't have an impact on the result but will cause wastage of calculation.

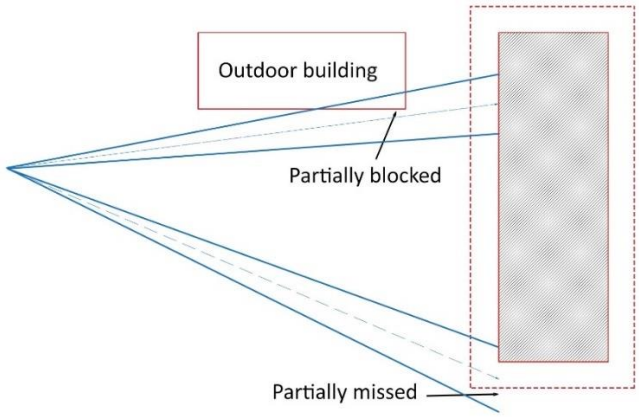


Figure 6-5 Refined tubes hitting and missing the marked building

Therefore, a step for capturing the refined tubes is needed after the generation, to detect the tubes that would hit the marked building among the generated candidates. The difficulty here is the arrangement of the processing pipeline. Firstly, the refined tubes and normal outdoor tubes exist at the same time. The outdoor tubes should be still collected as normal for the pathloss result, while the refined tubes are to be captured for re-launching. Mixing them would introduce excessive overhead to the existing pipeline as the number of refined tubes is significant according to Equation (6-1). Secondly, refined tubes still propagate in the outdoor scenario, so they are subjected to the intersection test from the same kernel. While one option is the duplication of the kernel, it will result in doubling the memory consumption, thus not preferred. Therefore, the single intersection kernel needs to be shared and scheduled for both outdoor tubes and refined tubes.

The original TLARS pipeline in Figure 5-12 is extended to support the process of refined tubes, as the MR-TLARS pipeline in Figure 6-6. Firstly, in the TLARS pipeline, the Intersection Processor, Ray Postprocessor and the New Ray Generator can all be encapsulated as the Core Processor. The Core Processor is activated by initial rays, then internally it generates subsequent rays from interactions. Meanwhile, all the finished rays

are exported to an external collector. In Figure 6-6, the top half branch is equivalent to the original TLARS pipeline. When the finished rays are exported for collection, they are also sent to the Multi-Resolution Trigger followed by the Multi-Resolution Generator, of which the output is the refined tubes. Then, these tubes are sent to the Multi-Resolution Core Processor, which is the same as the top Core Processor. Thus, its output will be the intersected refined tubes, which will be collected in the Multi-Resolution Ray Collector. In there, no pathloss result is generated, but the rays meeting the capturing condition will be cached.

In this case, the normal outdoor tubes and generated refined tubes are processed separately in the top and lower branch in Figure 6-6. Then internally, the Core Processor and Multi-Resolution Core Processor are delegating the intersection calculation to a shared Intersection Processor, which is not shown in the figure. In the shared Intersection Processor, the requests from the Multi-Resolution Core Processor are given with higher priority than from the Core Processor. This is because the refined tubes are the child of outdoor tubes, and the DFS algorithm should be enforced to give priority to the child tubes for less memory consumption.

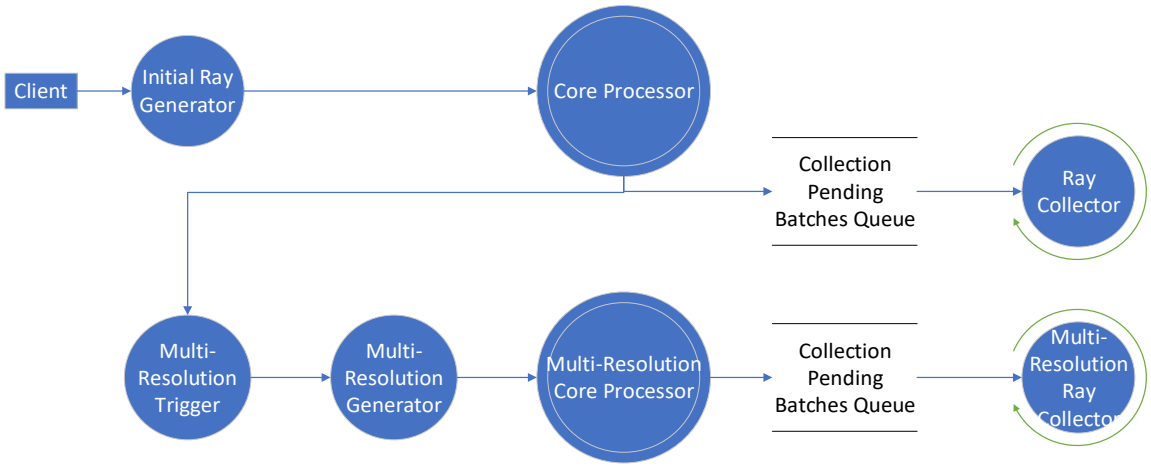


Figure 6-6 MR-TLARS modules and pipeline

6.1.4 Refined Tubes Re-launching

Once the refined tubes are captured in the outdoor scenario, they are subjected to be re-launched in the corresponding indoor scenario, in which more detailed on-site entities will be available, e.g. windows, doors and internal walls. Thereafter, the process of the tubes is no different from the calculation of a transmitter in an indoor scenario, thus the full original TLARS engine will be reused for the remaining calculation. The only additional step is the transformation of the coordinates, as outdoor scenario and indoor scenario are likely to use different local coordinate systems. The transformation is a combination of rotation and translation. Then for the fields in the structure `RayWithPayload` in Figure 5-11, they need to be applied with corresponding transformations to be continued in the indoor coordinate system.

The rotation usually happens on the horizontal xoy plane, as the outdoor and indoor coordinates are aligned with the sea level. Therefore, the affine rotation matrix from outdoor to indoor for a counter-clockwise angle θ to the x-axis can be given as [135]

$$\boldsymbol{R} = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-2}$$

And the translation matrix from outdoor to indoor can be given as

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-3}$$

where the v_x, v_y and v_z are the offsets on x, y and z axes. Then the points in the outdoor scenario, e.g. the `Origin` and `SourceLocation`, need to be applied with the composite transformation, $\boldsymbol{T} \times \boldsymbol{R}$. And the vectors in the outdoor scenario, e.g. the `Direction`,

`CrossSectionVector1` and `CrossSectionVector2`, need to be applied with the rotation matrix $\boldsymbol{R}$ only. For the rest fields, which are all scalar, no change is needed.

6.2 Indoor-to-Outdoor Radio Prediction with Extended Tube Launching

Another aspect of the hybrid calculation is the transition from indoor to outdoor. To the contrary of the O2I case, now the rays will start with high resolution in the indoor scenario. On leaving the scope of the indoor building to the outdoor area, a lower calculation resolution needs to be applied. Otherwise, the complexity would be equivalent to the calculation of an outdoor transmitter in an outdoor scenario but with a high resolution desired for an indoor scenario. In this case, the increased complexity may be hard to handle according to its dependency on the resolution. Also, the excessive high resolution may not introduce additional benefit given the limited resolution on the outdoor geometrical information itself [1]. Therefore, a downcast of the resolution is needed during the transition from indoor to outdoor.

The steps for the indoor-to-outdoor calculation is simpler than the outdoor-to-indoor, from which first two steps Generation Triggering and Casting are not needed since a high-resolution tube can be converted to a low-resolution tube by extending the tube length. So, the required remaining steps in I2O calculation are Extended Tubes Capturing and Extended Tubes Re-launching.

6.2.1 Extended Tubes Capturing and Re-launching

The indoor tube can be used as it is, while with some transformation applied for the outdoor calculation. But not all the indoor tubes could enter the outdoor area. The detection of a tube leaving the indoor scenario can be given by that no triangle intersection is detected

in the Intersection Processor. This is valid as a tube hitting a polygon must be still in the scenario, and the tube leaving the scenario must be from the transmission or reflection on the external polygon, from where no further polygon would exist.

Thus, once a tube without intersection is detected, it will be converted to an extended tube to be re-launched in the outdoor calculation. The conversion is given in Figure 6-7. For an indoor tube with a length T_{indoor} targeting at resolution R_{indoor}, the conversion is to extend the length of the tube so that the width of the cross-section in the end matches the outdoor resolution $R_{outdoor}$. According to the analogy, the length of the extended tube can be given as:

$$T_{outdoor} = \frac{R_{outdoor}}{R_{indoor}} \times T_{indoor} \tag{6-4}$$

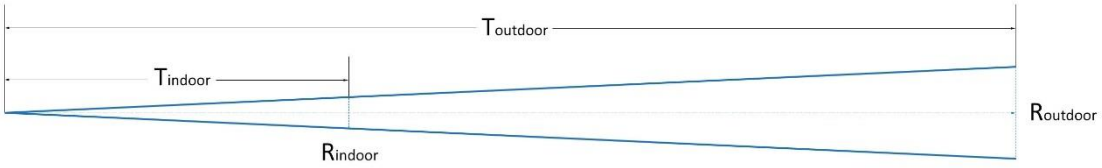


Figure 6-7 The conversion from an indoor tube to an extended outdoor tube

Therefore, the number of extended tubes will be equal to the number of indoor tubes exiting the scene. The complexity is also reduced since the extended tubes are adapted to the outdoor resolution. To perform the calculation, the same MR-TLARS design given in Figure 6-6 can also be used for the I2O case. But the bottom branch of the design is simplified to only check the rays exiting the indoor scenario and perform the extension of tube length, therefore, no additional Core Processor is needed.

Once the extended tubes are captured, they are subject to be re-launched in the outdoor scenario, which is similar to the process in Section 6.1.4. Therefore, the

transformations inverted from previous matrixes need to be applied to the extended tubes. Thereafter, the extended tubes will propagate in the outdoor scenario, as if they are emitted in the outdoor. Meanwhile, since these tube bears the interaction loss and travelled distance in the prior indoor scenario, their total loss will include the corresponding history from indoor. So, the I2O result can be fully represented by the proposed method.

6.3 MR-TLARS Results and Measurement Validation

In this section, the MR-TLARS will be used for simulating an O2I scenario. The performance will be checked, and the result will be validated against measurement captured in a corresponding indoor environment. Then the simulation will also be performed for the I2O calculation in the same scenario. But the result won't be validated due to the lack of measurement.

6.3.1 Taiyuan Scenario O2I Result

The modelled Taiyuan scenario is an urban area with a dimension of $4300m \times 3400m \times 145m$, and 87924 building polygons. No terrain data is included as the area is flat. The corresponding indoor scenario is a 6-level shopping mall with a dimension of $110m \times 59m \times 18m$ including 1631building polygons. The calculation uses $5m$ outdoor resolution and $0.5m$ indoor resolution, and the result is shown in Figure 6-8, where the location of the outdoor antenna and the indoor building is marked as well. The antenna is from an existing macro base station located $25m$ above the ground on the edge of another building. The transmitting frequency is 1900MHz and down tilt of -6 degree is applied to the antenna. The distance between the antenna and the surface of the indoor building is about $190m$.

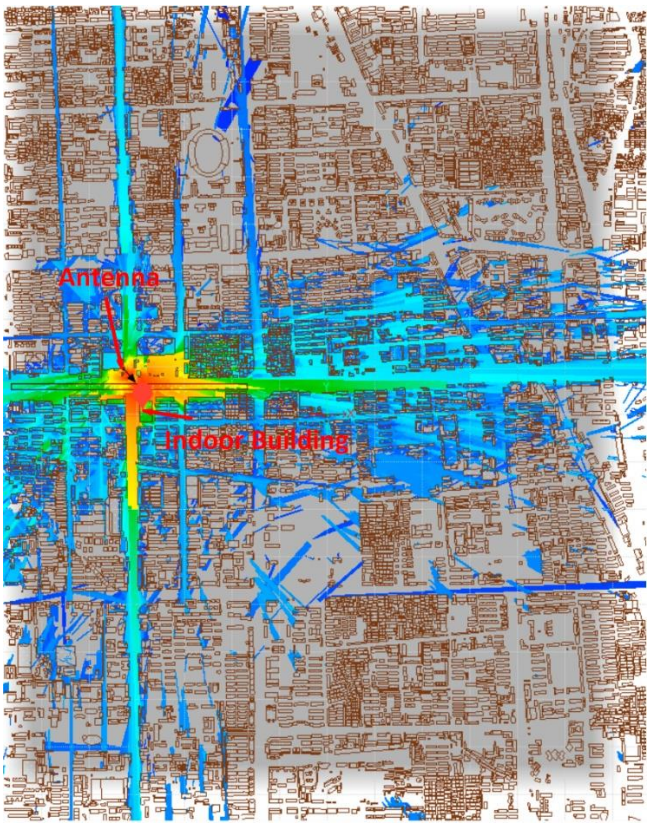


Figure 6-8 Taiyuan outdoor result

In Figure 6-9, a 3D view of the result presents the simulated signal radiating from the outdoor into the indoor area. In outdoor, the rough result at $5m$ resolution is provided, while in indoor, the fine result at $0.5m$ resolution is provided in a more detailed building model on each floor. The surface of the indoor building is made from glass, thus small transmission loss is introduced when the outdoor signal firstly enters the indoor area. Thereafter, the blocking and reflection on the indoor structures are presented in the indoor result.

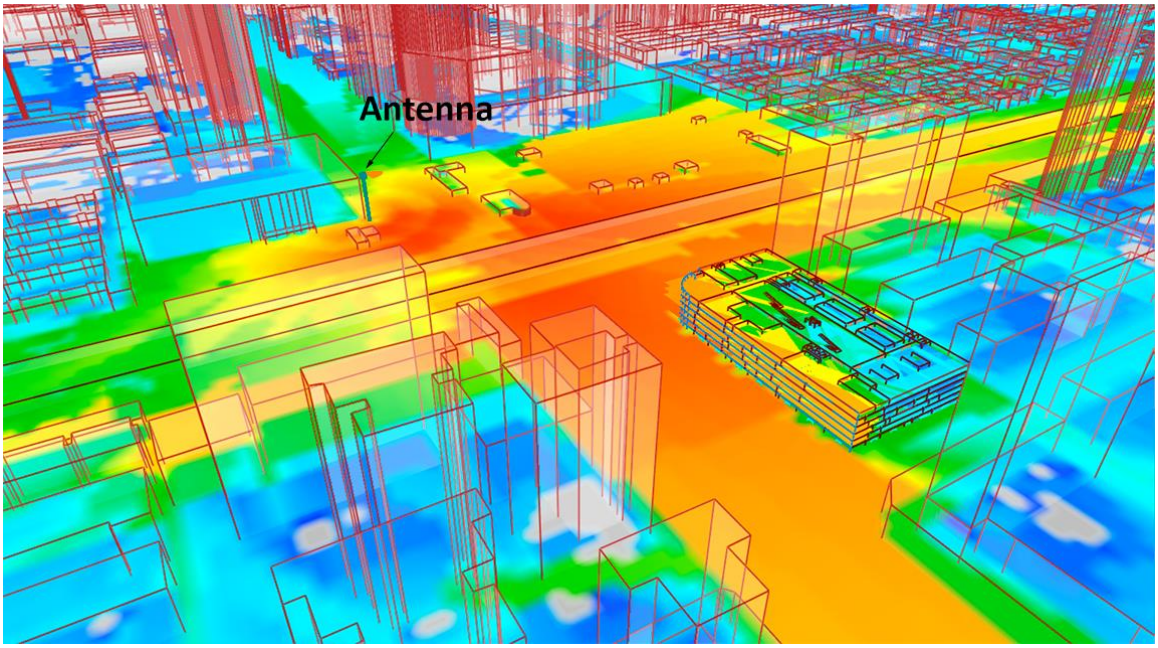


Figure 6-9 Taiyuan O2I result in 3D view

The calculation takes 22sec and 1872MB for the outdoor area, while in the indoor area 6.8sec and 568MB are spent. The time spent and memory consumptions for the outdoor are close to the Munich scenario, which is having a similar dimension and building density to the Taiyuan scenario. Meanwhile, compared to the much larger consumptions spent in Santiago scenario, it can be concluded that significant complexity is introduced from the extreme dense building polygons and terrain polygons. In terms of the additional O2I projection, no obvious overhead is experienced in the simulation, due to the efficient refined tube casting mechanism introduced in Section 6.1.2.

The measurement is a walk test on the 4^{th} floor of the shopping mall using a handheld device holding at $1.3m$ above the floor height, which is about $13.3m$ above the ground level. The comparison between the calculated pathloss values and the measurement is plotted in Figure 6-10. The outdoor signal is coming from the bottom left of the floor plan, where the measurement also shows the strongest value. Meanwhile, on the top branch

of measurement route, among the cyan spots, there are a few green spots, which are the result of reflection on external outdoor buildings, as can be seen in Figure 6-9. The calculated result does approximate radio paths contributing to those green spots on the top branch. While such an effect would be hard to be presented in an empirical model or a deterministic model using hybrid O2I transition, as the actual paths from outdoor to indoor may not be persisted.

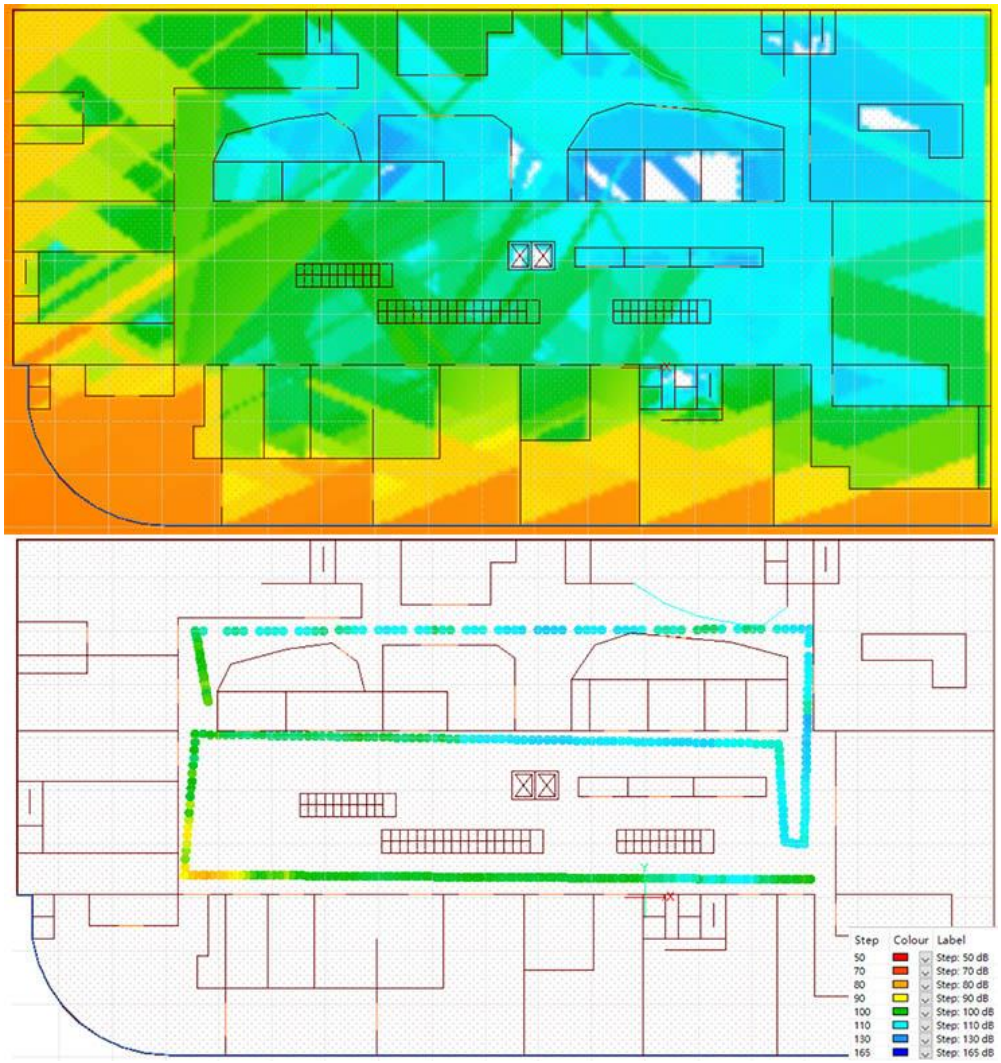


Figure 6-10 Taiyuan indoor result on the 4th floor compared with measurement

The comparison between the measured values and calculated results before and after calibration is given in Figure 6-11, and the histogram of the error between the calibration and measurement is given in Figure 6-12. Before the calibration, the ME is 7.57dB and the RMSE is 9.06dB. After calibration, the ME and RMSE are reduced to 5.13dB and 6.89dB. Thus, an accurate match is presented between the measurement and calculated result in the modelled O2I scenario.

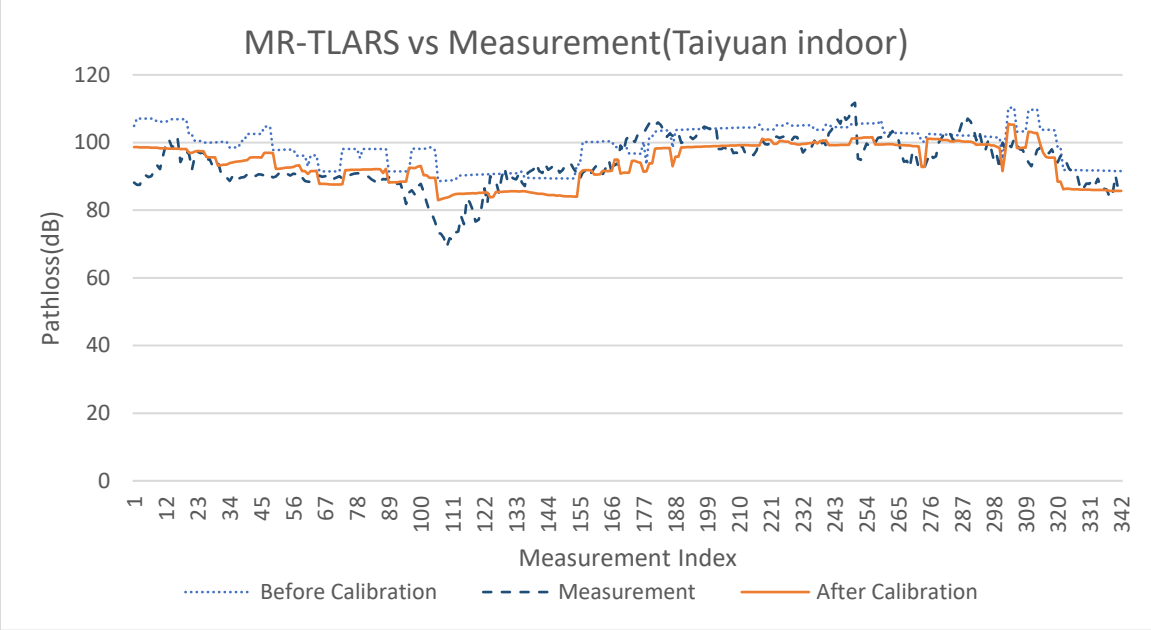


Figure 6-11 Taiyuan indoor measurement vs. MR-TLARS results before and after calibration

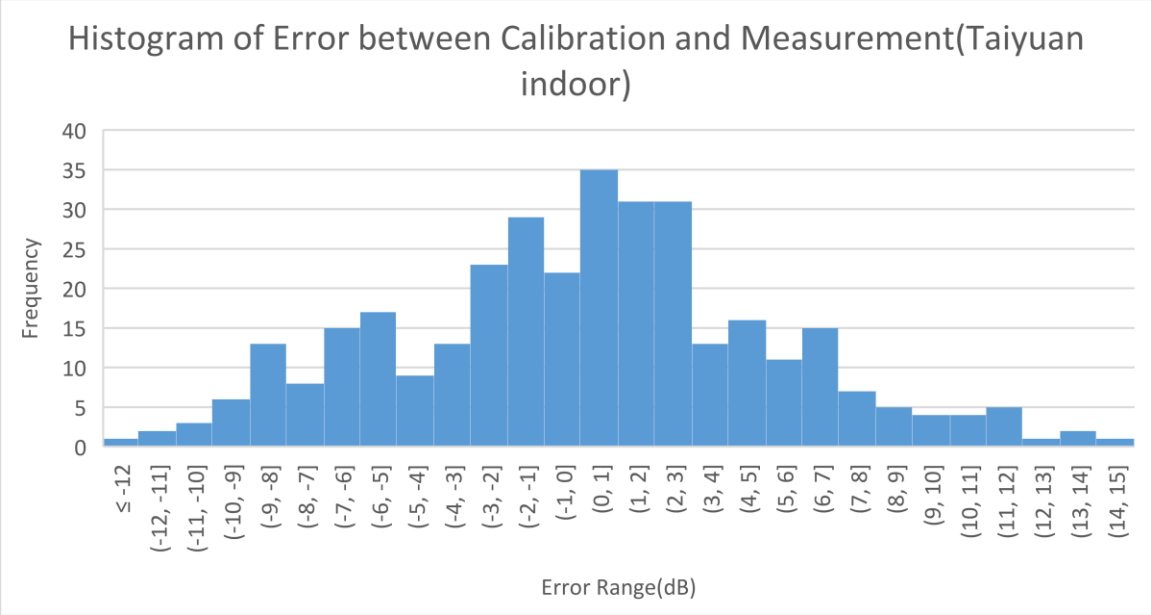
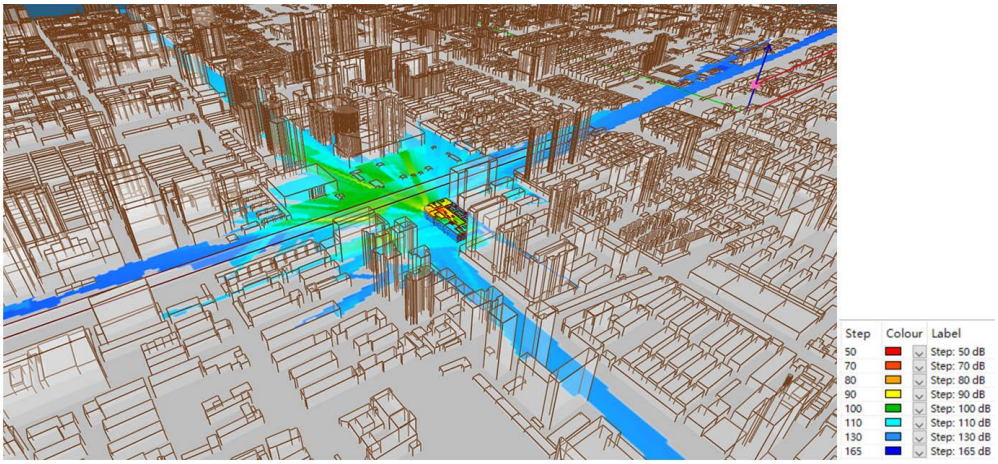


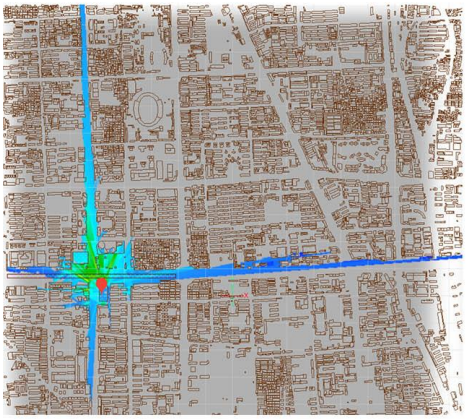
Figure 6-12 Histogram of error between the calibration and measurement in Taiyuan indoor

6.3.2 Taiyuan Scenario I2O Result

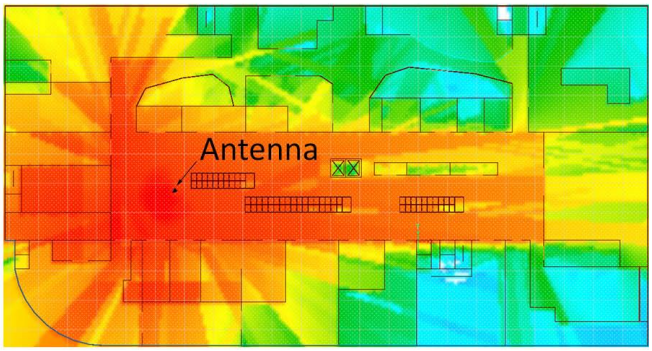
Then the I2O function of the MR-TLARS will be presented in the same Taiyuan scenario by placing an antenna in the indoor area. Due to the lack of measurement, only the simulation result and performance metrics will be given in this section. The antenna is located on the 4^{th} floor of the shopping mall at $2.8m$ above the floor height, as shown in Figure 6-13, where the simulation result is presented in different views as well. On the top 3D view, the coverage of the signal in the indoor area, and the propagation from indoor to the outdoor area are presented. Then on the bottom two images, the result in the outdoor area and indoor area are shown respectively. The outdoor result is shown at $1m$ above the ground and $13.8m$ below the antenna. Such variation in z is presented as the strength drop during the transition, e.g. the signal leaving the indoor boundary is orange coloured while the signal initialling in the outdoor area is green coloured.



I2O 3D view



Outdoor



Indoor 4th floor

Figure 6-13 Taiyuan I2O result in different views

Performance-wise, the indoor part of the calculation takes 8.9sec and 691MB memory, while the outdoor part takes 14.6sec and 1156MB memory. The I2O simulation has more indoor consumption and less outdoor consumption than the O2I simulation, but in overall the performances are similar. This is as expected as in the I2O, the indoor calculation not only calculates the indoor pathloss, but will also capture the rays leaving the indoor area, and vice versa for the outdoor calculation. The validation of the I2O result will be a future work of this study.

6.4 Conclusion

This chapter has proposed the O2I and I2O propagation model, MR-TLARS. Benefiting from the original adaptive splitting mechanism, the MR-TLARS is able to provide a deterministic transition between the outdoor and indoor scenarios seamless, which is not seen in the other proposed models in the literature. Then during the simulation, such a mechanism is shown with the advantage of representing specific indoor paths resulting from outdoor reflection. A summary of the metrics from the simulations in both O2I and I2O is listed in Table 6-1. Performance-wise, the time and memory consumptions in the outdoor part of the calculation are similar to the consumptions for stand-alone outdoor calculation performed in Section 5.6.3. The similarity presents the efficiency of the mechanism and module design of MR-TLARS in providing the O2I and I2O simulation.

	Taiyuan O2I	Taiyuan I2O
Time spent (outdoor)	22sec	14.6sec
Time spent (indoor)	6.8sec	8.9sec
Memory consumed (outdoor)	1872MB	1156MB
Memory consumed (indoor)	568MB	691MB
RMSE (before calibration)	9.06dB	NA
RMSE (after calibration)	6.89dB	NA

Table 6-1 Simulation results summary

7 Summary and Future Work

In this study, the problem of providing an efficient radio propagation model for full 3D volume prediction is identified. The demand for such model arises from the evolvement to the 5G technology, where UDN needs to be planned and the exact locations of massive small cells need to be determined through propagation model assisted simulations. While many challenges are faced including the angular dispersion, double-counting and indoor/outdoor transition etc., there lacks a comprehensive solution in existing studies to be efficiently applied in the described planning scenarios. Especially the desired model is a full 3D vector-based deterministic model, where the problem complexity is shown to be excessive.

Then through comparison among potential existing candidates, it is concluded that the ray-launching method should be preferred for solving the identified problem, while the other methods have their advantage in different domains. However, various challenges still exist for the ray-launching method. Among them, the most critical one is the increase of complexity magnitude by one degree due to the angular dispersion.

Although the adoption of hardware acceleration is showing great improvement in the calculation speed, most of the studies are still limiting the scope to a few reception points and having no consideration of angular dispersion. Therefore, it is of the interest of this study to test the actual performance improvement from hardware acceleration in a ray-launching radio model. During the implementation, many problems for the ray-based method are identified and discussed, including the ray-triangle intersection, ray-box intersection, grid traversal and spatial indexing etc. In the end, the prototype propagation model GORL is showing a massive speed improvement up to about 100 times from

hardware acceleration. However, when the angular dispersion is considered in a brute force method, over launching, the improvement delivered from hardware acceleration is still shown not enough to provide an efficient solution.

From the experience in the GORL, it is acknowledged that the complexity involved from angular dispersion needs to be solved in continuous modelling. Therefore, the idea of tube-launching is brought into consideration. While existing methods lack an efficient solution for the problems of tube intersection, tube splitting and tube sampling, the proposed innovative method TLARS approaches the tube adaptively using central ray splitting. Meanwhile, the mechanisms for multipath and diffraction are also introduced. In the simulation of TLARS in a few outdoor and indoor scenarios, the result and the performance do present TLARS as an efficient solution for radio prediction in 3D volume. Also, the validation against various measurements demonstrates the accuracy and versatility of TLARS.

Thereafter, the adaptive mechanism introduced in TLARS is extended to provide a multi-resolution solution for O2I and I2O scenarios. A seamless transition crossing the boundary between outdoor and indoor scenarios is provided by the proposed MR-TLARS in a deterministic manner. With an extended design from the TLARS module, the performance in the simulation within the hybrid Taiyuan scenario is showing efficient O2I and I2O calculation, as no obvious overhead is experienced compared to simulations in stand-alone scenarios. Also, the validation against an O2I measurement is showing the advantage of MR-TLARS in simulating the impact of the outdoor environment in the indoor result.

In the end, the TLARS and MR-TLARS are presented to be suitable for solving the identified problem, which is to provide full 3D volume radio prediction efficiently. However, future works are needed especially for the consideration of dynamic and small scale components in the environment, and other efficiency improvements.

Firstly, the application of hardware acceleration could be extended to more modules. So far, in the implementation of GORL and TLARS, the OpenCL kernels are created for the spatial indexing, intersection test, grid traversal and ray collection, where signification performance improvement is presented. However, there are still some heavy calculation tasks remaining executed on the CPU program, for example, the new ray generation, polygon hit splitting and central ray splitting. It is worth to offload these tasks to corresponding kernels for potential further improvement of the speed. Meanwhile, the current modelling of terrain data is not efficient and should be replaced with smooth terrain modelling for fewer triangles.

Secondly, the simulation of range beyond a few kilometres is becoming less efficient using a full deterministic approach. This is because the variation in the close environment is becoming less significant for long-range radio links, while the computational complexity has a dependency on the range by the power of 2. Thus, it is worth to identify a transition point in the range, beyond which the calculation method can be switched to an empirical method like the knife-edge. Such a hybrid model should benefit the goods from both worlds, that the efficiency can be improved for large scenarios with no obvious impact on the accuracy.

Thirdly, it is impractical to model the scattering using a pure ray-launching method, as the complexity of calculating scattering from a single ray on a polygon is approximating

the complexity of calculating a point source in the scenario. However, stochastic methods do provide efficient solutions for the scattering. While these methods assume the locations of the scatterers based pre-defined distributions, it is promising to improve the approximation of the distribution with the assistance of the rays collected in the deterministic method. Therefore, higher accuracy than the stochastic method with much less complexity than the deterministic method may be achieved for modelling scattering.

After all, like suggested in [6] that one future trend of the radio propagation model is the extension from the empirical methods with more deterministic approaches, the other aspect could be the extension of deterministic models using empirical approaches to form a comprehensive radio propagation model.

References

[1] P. Almers *et al.*, "Survey of channel and radio propagation models for wireless MIMO systems," *EURASIP Journal on Wireless Communications and Networking,* vol. 2007, no. 1, p. 019070, 2007.

[2] C. Phillips, D. Sicker, and D. Grunwald, "A survey of wireless path loss prediction and coverage mapping methods," *IEEE Communications Surveys & Tutorials,* vol. 15, no. 1, pp. 255-270, 2012.

[3] Y. Niu, Y. Li, D. Jin, L. Su, and A. V. Vasilakos, "A survey of millimeter wave communications (mmWave) for 5G: opportunities and challenges," *Wireless networks,* vol. 21, no. 8, pp. 2657-2676, 2015.

[4] T. K. Sarkar, Z. Ji, K. Kim, A. Medouri, and M. Salazar-Palma, "A survey of various propagation models for mobile communication," *IEEE Antennas and propagation Magazine,* vol. 45, no. 3, pp. 51-82, 2003.

[5] Z. Yun and M. F. Iskander, "Ray tracing for radio propagation modeling: Principles and applications," *IEEE Access,* vol. 3, pp. 1089-1100, 2015.

[6] C.-X. Wang, J. Bian, J. Sun, W. Zhang, and M. Zhang, "A survey of 5G channel measurements and models," *IEEE Communications Surveys & Tutorials,* vol. 20, no. 4, pp. 3142-3168, 2018.

[7] B. e. Gschwendtner, B. B. G. Wolfle, and F. M. Landstorfer, "Ray Tracing Vs. Ray Launching In 3-D Microcell Modelling," *Computer Science,* 1995.

[8] S. J. Flores, L. F. Mayorgas, and F. A. Jimenez, "Reception algorithms for ray launching modeling of indoor propagation," in *Proceedings RAWCON 98. 1998 IEEE Radio and Wireless Conference (Cat. No. 98EX194)*, 1998: IEEE, pp. 261-264.

[9] J. Tan, Z. Su, and Y. Long, "A Full 3-D GPU-based Beam-Tracing Method for Complex Indoor Environments Propagation Modeling," *IEEE Transactions on Antennas and Propagation,* vol. 63, no. 6, pp. 2705-2718, 2015, doi: 10.1109/TAP.2015.2415036.

[10] D. Shi, X. Tang, C. Wang, M. Zhao, and Y. Gao, "A GPU implementation of a Shooting and bouncing ray tracing method for radio wave propagation," *Applied Computational Electromagnetics Society Journal,* vol. 32, pp. 614-620, 07/01 2017.

[11] Z. Lai, H. Song, P. Wang, H. Mu, L. Wu, and J. Zhang, "Implementation and validation of a 2.5 d intelligent ray launching algorithm for large urban scenarios," in *2012 6th European Conference on Antennas and Propagation (EUCAP)*, 2012: IEEE, pp. 2396-2400.

[12] J.-M. Gorce, K. Jaffres-Runser, and G. De La Roche, "Deterministic approach for fast simulations of indoor radio wave propagation," *IEEE transactions on Antennas and Propagation,* vol. 55, no. 3, pp. 938-948, 2007.

[13] Y. Corre and Y. Lostanlen, "Three-dimensional urban EM wave propagation model for radio network planning and optimization over large areas," *IEEE Transactions on Vehicular Technology,* vol. 58, no. 7, pp. 3112-3123, 2009.

[14] X. Ge, S. Tu, G. Mao, C.-X. Wang, and T. Han, "5G ultra-dense cellular networks," *IEEE Wireless Communications,* vol. 23, no. 1, pp. 72-79, 2016.

[15] E. J. Oughton and Z. Frias, "The cost, coverage and rollout implications of 5G infrastructure in Britain," *Telecommunications Policy,* vol. 42, no. 8, pp. 636-652, 2018.
[16] E. Amaldi, A. Capone, and F. Malucelli, "Radio planning and coverage optimization of 3G cellular networks," *Wireless Networks,* vol. 14, no. 4, pp. 435-447, 2008.
[17] A. Valcarce and J. Zhang, "Empirical indoor-to-outdoor propagation model for residential areas at 0.9–3.5 GHz," *IEEE Antennas and Wireless Propagation Letters,* vol. 9, pp. 682-685, 2010.
[18] H. Okamoto, K. Kitao, and S. Ichitsubo, "Outdoor-to-indoor propagation loss prediction in 800-MHz to 8-GHz band for an urban area," *IEEE Transactions on vehicular Technology,* vol. 58, no. 3, pp. 1059-1067, 2008.
[19] C. Oestges and A. J. Paulraj, "Propagation into buildings for broad-band wireless access," *IEEE Transactions on Vehicular Technology,* vol. 53, no. 2, pp. 521-526, 2004.
[20] V. Nurmela *et al.*, "Deliverable D1. 4 METIS channel models," in *Proc. Mobile Wireless Commun. Enablers Inf. Soc.(METIS)*, 2015, p. 1.
[21] Y. Corre, Y. Lostanlen, and Y. Le Helloco, "A new approach for radio propagation modeling in urban environment: Knife-edge diffraction combined with 2D ray-tracing," in *Vehicular Technology Conference. IEEE 55th Vehicular Technology Conference. VTC Spring 2002 (Cat. No. 02CH37367)*, 2002, vol. 1: IEEE, pp. 507-511.
[22] K. Rizk, J.-F. Wagen, and F. Gardiol, "Two-dimensional ray-tracing modeling for propagation prediction in microcellular environments," *IEEE Transactions on Vehicular Technology,* vol. 46, no. 2, pp. 508-518, 1997.
[23] Y. Yuan, C.-X. Wang, X. Cheng, B. Ai, and D. I. Laurenson, "Novel 3D geometry-based stochastic models for non-isotropic MIMO vehicle-to-vehicle channels," *IEEE Transactions on Wireless Communications,* vol. 13, no. 1, pp. 298-309, 2013.
[24] Q. Zhu, Y. Wang, K. Jiang, X. Chen, W. Zhong, and N. Ahmed, "3D non-stationary geometry-based multi-input multi-output channel model for UAV-ground communication systems," *IET Microwaves, Antennas & Propagation,* vol. 13, no. 8, pp. 1104-1112, 2019.
[25] A. Abdi and M. Kaveh, "A space-time correlation model for multielement antenna systems in mobile fading channels," *IEEE Journal on Selected Areas in communications,* vol. 20, no. 3, pp. 550-560, 2002.
[26] X. Yin and X. Cheng, *Propagation channel characterization, parameter estimation, and modeling for wireless communications*. John Wiley & Sons, 2016.
[27] J. Karedal *et al.*, "A geometry-based stochastic MIMO model for vehicle-to-vehicle communications," *IEEE Transactions on Wireless Communications,* vol. 8, no. 7, pp. 3646-3657, 2009.
[28] C. S. Patel, G. L. Stuber, and T. G. Pratt, "Simulation of Rayleigh-faded mobile-to-mobile communication channels," *IEEE Transactions on Communications,* vol. 53, no. 11, pp. 1876-1884, 2005.

[29] J. W. Wallace and M. A. Jensen, "Modeling the indoor MIMO wireless channel," *IEEE Transactions on Antennas and Propagation*, vol. 50, no. 5, pp. 591-599, 2002.
[30] J. W. Wallace, M. A. Jensen, A. L. Swindlehurst, and B. D. Jeffs, "Experimental characterization of the MIMO wireless channel: Data acquisition and analysis," *IEEE Transactions on Wireless Communications*, vol. 2, no. 2, pp. 335-343, 2003.
[31] V. Erceg, "IEEE P802. 11 wireless LANs TGn channel models," *IEEE 802.11-03/940r4*, 2004.
[32] G. Acosta-Marum and M. A. Ingram, "Six time-and frequency-selective empirical channel models for vehicular wireless LANs," *IEEE Vehicular Technology Magazine*, vol. 2, no. 4, pp. 4-11, 2007.
[33] J. Schelleng, C. Burrows, and E. Ferrell, "Ultra-short-wave propagation," *Proceedings of the Institute of Radio Engineers*, vol. 21, no. 3, pp. 427-463, 1933.
[34] G. Millington, R. Hewitt, and F. S. Immirzi, "Double knife-edge diffraction in field-strength predictions," *Proceedings of the IEE - Part C: Monographs*, vol. 109, no. 16, pp. 419-429, 1962, doi: 10.1049/pi-c.1962.0059.
[35] J. Deygout, "Multiple knife-edge diffraction of microwaves," *IEEE Transactions on Antennas and Propagation*, vol. 14, no. 4, pp. 480-489, 1966.
[36] R. Luebbers, "Finite conductivity uniform GTD versus knife edge diffraction in prediction of propagation path loss," *IEEE Transactions on Antennas and Propagation*, vol. 32, no. 1, pp. 70-76, 1984.
[37] F. Dickson, J. Egli, J. Herbstreit, and G. Wickizer, "Large reductions of vhf transmission loss and fading by the presence of a mountain obstacle in beyond-line-of-sight paths," *Proceedings of the IRE*, vol. 41, no. 8, pp. 967-969, 1953.
[38] R. Kirby, H. Dougherty, and P. McQuate, "Obstacle gain measurements over pikes peak at 60 to 1,046 Mc," *Proceedings of the IRE*, vol. 43, no. 10, pp. 1467-1472, 1955.
[39] I. Union, "Propagation by diffraction," in "Recommendation ITU-R P.526-14," tech. rep., Telecommunication standardization sector of ITU (ITU-T), 2018.
[40] Y. Corre and Y. Lostanlen, "3D urban propagation model for large ray-tracing computation," in *2007 International Conference on Electromagnetics in Advanced Applications*, 2007: IEEE, pp. 399-402.
[41] E. Damosso and L. M. Correia, "COST action 231: Digital mobile radio towards future generation systems: Final report," *European commission*, 1999.
[42] M. Lott and I. Forkel, "A multi-wall-and-floor model for indoor radio propagation," in *IEEE VTS 53rd Vehicular Technology Conference, Spring 2001. Proceedings (Cat. No. 01CH37202)*, 2001, vol. 1: IEEE, pp. 464-468.
[43] L. Li, Y. Ibdah, Y. Ding, H. Eghbali, S. H. Muhaidat, and X. Ma, "Indoor multi-wall path loss model at 1.93 GHz," in *MILCOM 2013-2013 IEEE Military Communications Conference*, 2013: IEEE, pp. 1233-1237.
[44] S. Hosseinzadeh, H. Larijani, K. Curtis, A. Wixted, and A. Amini, "Empirical propagation performance evaluation of LoRa for indoor environment," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, 2017: IEEE, pp. 26-31.

[45] K. Hari, D. Baum, A. Rustako, R. Roman, and D. Trinkwon, "Channel models for fixed wireless applications," *IEEE 802.16 Broadband wireless access working group*, 2003.
[46] M. Hata, "Empirical formula for propagation loss in land mobile radio services," *IEEE transactions on Vehicular Technology*, vol. 29, no. 3, pp. 317-325, 1980.
[47] E. C. Committee, "Within the European conference of postal and telecommunications administration (CEPT),"the analysis of the coexistence of FWA cells in the 3.4–3.8 GHz band"," *ECC Report*, vol. 33, 2003.
[48] M. S. Mollel and M. Kisangiri, "Comparison of empirical propagation path loss models for mobile communication," 2014.
[49] V. Abhayawardhana, I. Wassell, D. Crosby, M. Sellars, and M. Brown, "Comparison of empirical propagation path loss models for fixed wireless access systems," in *2005 IEEE 61st Vehicular Technology Conference*, 2005, vol. 1: IEEE, pp. 73-77.
[50] G. R. A. N. W. Group, "Study on channel model for frequencies from 0.5 to 100 GHz (Release 14)," 3GPP TR 38.901, 2017.
[51] A. Maltsev, A. Pudeyev, and Y. Gagiev, "Channel Models for IEEE 802.11 ay," *IEEE doc*, pp. 802.11-15, 2017.
[52] A. Maltsev, A. Pudeyev, A. Lomayev, and I. Bolotin, "Channel modeling in the next generation mmWave Wi-Fi: IEEE 802.11 ay standard," in *European Wireless 2016; 22th European Wireless Conference*, 2016: VDE, pp. 1-8.
[53] K. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Transactions on antennas and propagation*, vol. 14, no. 3, pp. 302-307, 1966.
[54] A. Taflove and S. C. Hagness, *Computational electrodynamics: the finite-difference time-domain method*. Artech house, 2005.
[55] A. Laner, A. Bahr, and I. Wolff, "FDTD simulations of indoor propagation," in *Proceedings of IEEE Vehicular Technology Conference (VTC)*, 1994: IEEE, pp. 883-886.
[56] Y. Wang, S. Safavi-Naeini, and S. K. Chaudhuri, "A hybrid technique based on combining ray tracing and FDTD methods for site-specific modeling of indoor radio wave propagation," *IEEE Transactions on antennas and propagation*, vol. 48, no. 5, pp. 743-754, 2000.
[57] J. Li, J.-F. Wagen, and E. Lachat, "Effects of large grid size discretization on coverage prediction using the ParFlow method," in *Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (Cat. No. 98TH8361)*, 1998, vol. 2: IEEE, pp. 879-883.
[58] A. Valcarce, G. De La Roche, Á. Jüttner, D. López-Pérez, and J. Zhang, "Applying FDTD to the coverage prediction of WiMAX femtocells," *EURASIP Journal on Wireless Communications and Networking*, vol. 2009, no. 1, p. 308606, 2009.
[59] A. Valcarce, G. De La Roche, and J. Zhang, "A GPU approach to FDTD for radio coverage prediction," in *2008 11th IEEE Singapore International Conference on Communication Systems*, 2008: IEEE, pp. 1585-1590.
[60] S. Akhtarzad and P. Johns, "Solution of Maxwell's equations in three space dimensions and time by the tlm method of numerical analysis," in *Proceedings of*

the Institution of Electrical Engineers, 1975, vol. 122, no. 12: IET, pp. 1344-1348.

[61] P. B. Johns, "On the Relationship Between TLM and Finite-Difference Methods for Maxwell's Equations (Short Paper)," *IEEE transactions on microwave theory and techniques,* vol. 35, no. 1, pp. 60-61, 1987.

[62] J.-M. Gorce, E. Jullo, and K. Runser, "An adaptive multi-resolution algorithm for 2D simulations of indoor propagation," 2003.

[63] C.-F. Yang, B.-C. Wu, and C.-J. Ko, "A ray-tracing method for modeling indoor wave propagation and penetration," *IEEE transactions on Antennas and Propagation,* vol. 46, no. 6, pp. 907-919, 1998.

[64] P. Meissner *et al.*, "On the use of ray tracing for performance prediction of UWB indoor localization systems," in *2013 IEEE International Conference on Communications Workshops (ICC)*, 2013: IEEE, pp. 68-73.

[65] U. S. Inan and A. S. Inan, *Electromagnetic Waves*. Prentice Hall, 2000.

[66] D. McNamara, C. Pistorius, and J. Malherbe, "The Uniform Geometrical theory of diffraction," *Artech House, London,* 1990.

[67] R. G. Kouyoumjian and P. H. Pathak, "A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface," *Proceedings of the IEEE,* vol. 62, no. 11, pp. 1448-1461, 1974.

[68] K. A. Remley, H. R. Anderson, and A. Weisshar, "Improving the accuracy of ray-tracing techniques for indoor propagation modeling," *IEEE transactions on vehicular technology,* vol. 49, no. 6, pp. 2350-2358, 2000.

[69] R. Valenzuela, "A ray tracing approach to predicting indoor wireless transmission," in *IEEE 43rd vehicular technology conference*, 1993: IEEE, pp. 214-218.

[70] J. W. McKown and R. L. Hamilton, "Ray tracing as a design tool for radio networks," *IEEE Network,* vol. 5, no. 6, pp. 27-30, 1991.

[71] M. Sanchez, L. De Haro, A. Pino, and M. Calvo, "Exhaustive ray tracing algorithm for microcellular propagation prediction models," *Electronics letters,* vol. 32, no. 7, pp. 624-625, 1996.

[72] S. Coco, A. Laudani, and L. Mazzurco, "A novel 2-D ray tracing procedure for the localization of EM field sources in urban environment," *IEEE transactions on magnetics,* vol. 40, no. 2, pp. 1132-1135, 2004.

[73] H.-W. Son and N.-H. Myung, "A deterministic ray tube method for microcellular wave propagation prediction model," *IEEE Transactions on Antennas and Propagation,* vol. 47, no. 8, pp. 1344-1350, 1999.

[74] D. N. Schettino, F. J. Moreira, and C. G. Rego, "Efficient ray tracing for radio channel characterization of urban scenarios," *IEEE Transactions on Magnetics,* vol. 43, no. 4, pp. 1305-1308, 2007.

[75] S. Grubisic, W. Carpes, J. Bastos, and G. Santos, "Association of a PSO optimizer with a quasi-3D ray-tracing propagation model for mono and multi-criterion antenna positioning in indoor environments," *IEEE transactions on magnetics,* vol. 49, no. 5, pp. 1645-1648, 2013.

[76] Z. Yun, S. Lim, and M. F. Iskander, "An integrated method of ray tracing and genetic algorithm for optimizing coverage in indoor wireless networks," *IEEE Antennas and Wireless Propagation Letters,* vol. 7, pp. 145-148, 2008.

[77] S. Y. Seidel and T. S. Rappaport, "Site-specific propagation prediction for wireless in-building personal communication system design," *IEEE transactions on Vehicular Technology*, vol. 43, no. 4, pp. 879-891, 1994.
[78] Z. Lai *et al.*, "Intelligent Ray Launching Algorithm for Indoor Scenarios," *Radioengineering*, vol. 20, no. 2, 2011.
[79] M. Zhu, A. Singh, and F. Tufvesson, "Measurement based ray launching for analysis of outdoor propagation," in *2012 6th European Conference on Antennas and Propagation (EUCAP)*, 2012: IEEE, pp. 3332-3336.
[80] D. M. Rose, S. Rey, and T. Kürner, "Differential 3D ray-launching using arbitrary polygonal shapes in time-variant indoor scenarios," in *2016 Global Symposium on Millimeter Waves (GSMM) & ESA Workshop on Millimetre-Wave Technology and Applications*, 2016: IEEE, pp. 1-4.
[81] D. Guevara, B. Medina, and A. Navarro, "Scattering Model for Ray Launching Tool, and Validation in 5.4 GHz Indoor," in *2018 IEEE-APS Topical Conference on Antennas and Propagation in Wireless Communications (APWC)*, 2018: IEEE, pp. 854-857.
[82] S. Tan and H. Tan, "A microcellular communications propagation model based on the uniform theory of diffraction and multiple image theory," *IEEE Transactions on Antennas and Propagation*, vol. 44, no. 10, pp. 1317-1326, 1996.
[83] J. G. Cleary and G. Wyvill, "Analysis of an algorithm for fast ray tracing using uniform space subdivision," *The Visual Computer*, vol. 4, no. 2, pp. 65-83, 1988.
[84] I. Wald and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in O (N log N)," in *2006 IEEE Symposium on Interactive Ray Tracing*, 2006: IEEE, pp. 61-69.
[85] I. Wald, "Realtime ray tracing and interactive global illumination," *Ausgezeichnete Informatikdissertationen 2004*, 2005.
[86] J. R. Shewchuk, "Triangle: A two-dimensional quality mesh generator and Delaunay triangulator," *Computer Science Division, University of California at Berkeley, Berkeley, California*, pp. 94720-1776, 2005.
[87] R. Mathar, M. Reyer, and M. Schmeink, "A cube oriented ray launching algorithm for 3D urban field strength prediction," in *2007 IEEE International Conference on Communications*, 2007: IEEE, pp. 5034-5039.
[88] G. Liang and H. L. Bertoni, "A new approach to 3-D ray tracing for propagation prediction in cities," *IEEE Transactions on Antennas and Propagation*, vol. 46, no. 6, pp. 853-863, 1998, doi: 10.1109/8.686774.
[89] K. Rizk, R. Valenzuela, S. Fortune, D. Chizhik, and F. Gardiol, "Lateral, full-3D and vertical plane propagation in microcells and small cells," in *VTC'98. 48th IEEE Vehicular Technology Conference. Pathway to Global Wireless Revolution (Cat. No. 98CH36151)*, 1998, vol. 2: IEEE, pp. 998-1003.
[90] M. F. Catedra, J. Perez, F. S. De Adana, and O. Gutierrez, "Efficient ray-tracing techniques for three-dimensional analyses of propagation in mobile communications: application to picocell and microcell scenarios," *IEEE Antennas and Propagation Magazine*, vol. 40, no. 2, pp. 15-28, 1998.
[91] H. Wang and T. S. Rappaport, "A parametric formulation of the UTD diffraction coefficient for real-time propagation prediction modeling," *IEEE Antennas and Wireless Propagation Letters*, vol. 4, pp. 253-257, 2005.

[92] J.-V. Rodríguez, J.-M. Molina-García-Pardo, and L. Juan-Llácer, "An improved solution expressed in terms of UTD coefficients for the multiple-building diffraction of plane waves," *IEEE Antennas and Wireless Propagation Letters,* vol. 4, pp. 16-19, 2005.
[93] A. V. Osipov, "A simple approximation of the Maliuzhinets function for describing wedge diffraction," *IEEE transactions on antennas and propagation,* vol. 53, no. 8, pp. 2773-2776, 2005.
[94] G. D. Durgin, "The practical behavior of various edge-diffraction formulas," *IEEE Antennas and Propagation Magazine,* vol. 51, no. 3, pp. 24-35, 2009.
[95] D. Umansky, G. De La Roche, Z. Lai, G. Villemaud, J.-M. Gorce, and J. Zhang, "A new deterministic hybrid model for indoor-to-outdoor radio coverage prediction," in *Proceedings of the 5th European Conference on Antennas and Propagation (EUCAP)*, 2011: IEEE, pp. 3615-3618.
[96] B. Xia, Z. Lai, G. Villemaud, and J. Zhang, "Joint ray launching method for outdoor to indoor propagation prediction based on interpolation," in *2015 9th European Conference on Antennas and Propagation (EuCAP)*, 2015: IEEE, pp. 1-5.
[97] T. Imai and Y. Okumura, "Study on hybrid method of ray-tracing and physical optics for outdoor-to-indoor propagation channel prediction," in *2014 IEEE International Workshop on Electromagnetics (iWEM)*, 2014: IEEE, pp. 249-250.
[98] A. Schmitz, T. Rick, T. Karolski, L. Kobbelt, and T. Kuhlen, "Beam tracing for multipath propagation in urban environments," in *2009 3rd European Conference on Antennas and Propagation*, 2009: IEEE, pp. 2631-2635.
[99] A. Schmitz, T. Rick, T. Karolski, T. Kuhlen, and L. Kobbelt, "Efficient rasterization for outdoor radio wave propagation," *IEEE transactions on visualization and computer graphics,* vol. 17, no. 2, pp. 159-170, 2010.
[100] B. R. Epstein and D. L. Rhodes, "GPU-accelerated ray tracing for electromagnetic propagation analysis," in *2010 IEEE International Conference on Wireless Information Technology and Systems*, 2010: IEEE, pp. 1-4.
[101] R. Felbecker, L. Raschkowski, W. Keusgen, and M. Peter, "Electromagnetic wave propagation in the millimeter wave band using the NVIDIA OptiX GPU ray tracing engine," in *2012 6th European Conference on Antennas and Propagation (EUCAP)*, 26-30 March 2012 2012, pp. 488-492, doi: 10.1109/EuCAP.2012.6206198.
[102] A. Navarro and D. Guevara, "Applicability of Game Engine for Ray Tracing Techniques in a Complex Urban Environment," in *2010 IEEE 72nd Vehicular Technology Conference - Fall*, 6-9 Sept. 2010 2010, pp. 1-5, doi: 10.1109/VETECF.2010.5594343.
[103] M. Schiller, A. Knoll, M. Mocker, and T. Eibert, "GPU accelerated ray launching for high-fidelity virtual test drives of VANET applications," in *2015 International Conference on High Performance Computing & Simulation (HPCS)*, 20-24 July 2015 2015, pp. 262-268, doi: 10.1109/HPCSim.2015.7237048.
[104] S. G. Parker *et al.*, "OptiX: a general purpose ray tracing engine," *ACM Trans. Graph.,* vol. 29, no. 4, pp. 1-13, 2010, doi: 10.1145/1778765.1778803.

[105] M. Reyer, T. Rick, and R. Mathar, "Hardware acceleration techniques for 3D urban field strength prediction," in *Proc. WFMN07*, Chemnitz, Germany, 12/26 2007.
[106] Z. Lai, N. Bessis, H. Song, J. Zhang, and G. Clapworthy, "An intelligent ray launching for urban prediction," in *2009 3rd European Conference on Antennas and Propagation*, 2009: IEEE, pp. 2867-2871.
[107] J. Liu, J. Wu, J. Chen, P. Wang, and J. Zhang, "Radio resource allocation in buildings with dense femtocell deployment," in *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, 2012: IEEE, pp. 1-5.
[108] L. Nagy, R. Dady, and A. Farkasvolgyi, "Algorithmic complexity of FDTD and ray tracing method for indoor propagation modelling," in *2009 3rd European Conference on Antennas and Propagation*, Berlin, Germany, 23-27 March 2009.
[109] W. J. Hoefer, "The transmission-line matrix method-theory and applications," *IEEE Transactions on Microwave Theory and Techniques,* vol. 33, no. 10, pp. 882-893, 1985.
[110] Z. Lai, N. Bessis, G. De La Roche, P. Kuonen, J. Zhang, and G. Clapworthy, "A new approach to solve angular dispersion of discrete ray launching for urban scenarios," in *2009 Loughborough Antennas & Propagation Conference*, 2009: IEEE, pp. 133-136.
[111] Rohde&Schwarz. "R&S®FSH handheld spectrum analyzer." https://www.rohde-schwarz.com/uk/product/fsh-productstartpage_63493-8180.html (accessed 2020).
[112] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.
[113] E. Papkelis *et al.*, "A radio-coverage prediction model in wireless communication systems based on physical optics and the physical theory of diffraction [Wireless Corner]," *IEEE Antennas and Propagation Magazine*, vol. 49, no. 2, pp. 156-165, 2007.
[114] L. Azpilicueta, F. Falcone, and R. Janaswamy, "A hybrid ray launching-diffusion equation approach for propagation prediction in complex indoor environments," *IEEE Antennas and Wireless Propagation Letters,* vol. 16, pp. 214-217, 2016.
[115] E. M. Vitucci *et al.*, "Tuning Ray Tracing for Mm‐wave Coverage Prediction in Outdoor Urban Scenarios," *Radio Science*, vol. 54, no. 11, pp. 1112-1128, 2019.
[116] Z. Dai and R. J. Watson, "Accelerating a ray launching model using GPU with CUDA," 2018.
[117] I. Ozimek, A. Hrovat, A. Vilhar, and T. JAVORNIK, "Parallel GPU Processing for Fast Radio Signal Propagation Computation in GRASS-RaPlaT," 2014.
[118] NVIDIA, *NVIDIA CUDA Programming Guide 2.0*. NVIDIA Cooperation, 2008.
[119] V. Thambawita, R. Ragel, and D. Elkaduwe, "To Use or Not to Use: Graphics Processing Units for Pattern Matching Algorithms," 12/25 2014.
[120] M. J. Harvey and G. De Fabritiis, "Swan: A tool for porting CUDA programs to OpenCL," *Computer Physics Communications,* vol. 182, no. 4, pp. 1093-1099, 2011/04/01/ 2011, doi: https://doi.org/10.1016/j.cpc.2010.12.052.
[121] K. Group. "OpenCL Overview." https://www.khronos.org/opencl/ (accessed 10/11/2019, 2019).

[122] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi, "Evaluating performance and portability of OpenCL programs," presented at the Workshop on Automatic Performance Tuning, 01/01, 2010.
[123] J. Fang, A. L. Varbanescu, and H. Sips, "A Comprehensive Performance Comparison of CUDA and OpenCL," in *2011 International Conference on Parallel Processing*, 13-16 Sept. 2011 2011, pp. 216-225, doi: 10.1109/ICPP.2011.45.
[124] G. Martinez, M. Gardner, and W. Feng, "CU2CL: A CUDA-to-OpenCL Translator for Multi- and Many-Core Architectures," in *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, 7-9 Dec. 2011 2011, pp. 300-307, doi: 10.1109/ICPADS.2011.48.
[125] A. Munshi, *OpenCL Programming Guide*. Addison-Wesley Professiona, 2011.
[126] M. Scarpino, *OpenCL in Action How to accelerate graphics and computations*. Manning Publications, 2011.
[127] R. Sotomayor, L. M. Sanchez, J. G. Blas, J. Fernandez, and J. D. Garcia, "Automatic CPU/GPU Generation of Multi-versioned OpenCL Kernels for C++ Scientific Applications," *Int. J. Parallel Program.*, vol. 45, no. 2, pp. 262-282, 2017, doi: 10.1007/s10766-016-0425-6.
[128] J. E. Greivenkamp, *Field Guide to Geometrical Optics*. SPIE PRESS, 2004.
[129] "IEEE standard definitions of terms for antennas," *IEEE Transactions on Antennas and Propagation*, vol. 17, no. 3, pp. 262-269, 1969, doi: 10.1109/TAP.1969.1139442.
[130] WIFEEB, "D1.1 Database on typical building materials wireless efficiency and energy efficiency," 2013.
[131] J. Jimenez-Delgado, C. Ogáyar, J. Noguera, and F. Paulano, *Performance Analysis for GPU-based Ray-triangle Algorithms*. 2014.
[132] T. M, #246, ller, and B. Trumbore, "Fast, minimum storage ray/triangle intersection," presented at the ACM SIGGRAPH 2005 Courses, Los Angeles, California, 2005.
[133] *PolyPartition*. (2018). [Online]. Available: https://github.com/ivanfratric/polypartition
[134] D. Eberly. "Triangulation by Ear Clipping." Geometric Tools, LLC. http://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf (accessed 21/11/2019, 2019).
[135] J. D. Foley *et al.*, *Computer graphics: principles and practice*. Addison-Wesley Professional, 1996.
[136] A. a. T. Lira dos Santos, Veronica and Lindoso, Jorge, "Review and Comparative Study of Ray Traversal Algorithms on a Modern GPU Architecture," in *WSCG 2014 Conference on Computer Graphics, Visualization and Computer Vision*, 2014, pp. 203-212. [Online]. Available: http://wscg.zcu.cz/WSCG2013/!_2013-WSCG-Full-proceedings.pdf
[137] I. Wald, S. Boulos, and P. Shirley, "Ray tracing deformable scenes using dynamic bounding volume hierarchies," *ACM Trans. Graph.*, vol. 26, no. 1, p. 6, 2007, doi: 10.1145/1189762.1206075.

[138] T. Karras. "Thinking Parallel, Part II: Tree Traversal on the GPU." https://devblogs.nvidia.com/thinking-parallel-part-ii-tree-traversal-gpu/ (accessed 27/11/2019, 2019).
[139] B. Smits, "Efficiency Issues for Ray Tracing," *Journal of Graphics Tools*, vol. 3, no. 2, pp. 1-14, 1998/01/01 1998, doi: 10.1080/10867651.1998.10487488.
[140] A. Williams, S. Barrus, R. K. Morley, and P. Shirley, "An Efficient and Robust Ray-Box Intersection Algorithm," *Journal of Graphics Tools*, vol. 10, no. 1, pp. 49-54, 2005/01/01 2005, doi: 10.1080/2151237X.2005.10129188.
[141] B. Cosenza, "A Survey on Exploiting Grids for Ray Tracing," in *Eurographics Italian Chapter Conference (2008)*, R. D. C. Vittorio Scarano, Ugo Erra, Ed., 2008: The Eurographics Association.
[142] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *Eurographics*, 1987, vol. 87, no. 3, pp. 3-10.
[143] I. Klyuzhin. "Fast raytracing through a 3D grid." https://www.mathworks.com/matlabcentral/fileexchange/56527-fast-raytracing-through-a-3d-grid (accessed 25/11/2019, 2019).
[144] D. Shi, N. Lv, and Y. Gao, "A diffraction ray tracing method based on curved surface ray tube for complex environment," *Applied Computational Electromagnetics Society Journal*, vol. 32, pp. 608-613, 07/01 2017.
[145] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive Indirect Illumination Using Voxel Cone Tracing," *Comput. Graph. Forum*, vol. 30, pp. 1921-1930, 09/01 2011, doi: 10.1111/j.1467-8659.2011.02063.x.
[146] G. Durgin, N. Patwari, and T. S. Rappaport, "Improved 3D ray launching method for wireless propagation prediction," *Electronics Letters*, vol. 33, no. 16, pp. 1412-1413, 1997.
[147] Y. Zhengqing, M. F. Iskander, and Z. Zhijun, "Development of a new shooting-and-bouncing ray (SBR) tracing method that avoids ray double counting," in *IEEE Antennas and Propagation Society International Symposium. 2001 Digest. Held in conjunction with: USNC/URSI National Radio Science Meeting (Cat. No.01CH37229)*, 8-13 July 2001 2001, vol. 1, pp. 464-467 vol.1, doi: 10.1109/APS.2001.958892.
[148] J. Zhong, L. Bin-Hong, W. Hao-Xing, C. Hsing-Yi, and T. K. Sarkar, "Efficient ray-tracing methods for propagation prediction for indoor wireless communications," *IEEE Antennas and Propagation Magazine*, vol. 43, no. 2, pp. 41-49, 2001, doi: 10.1109/74.924603.
[149] J. Richter, M. O. Al-Nuaimi, and L. P. Ivrissimtzis, "Optimisation of radio coverage in urban microcells using a UTD based ray-tracing model," *IEE Proceedings - Microwaves, Antennas and Propagation*, vol. 151, no. 3, pp. 187-192, 2004, doi: 10.1049/ip-map:20040272.
[150] S. Saunders and F. Bonar, "Explicit multiple building diffraction attenuation function for mobile radio wave propagation," *Electronics Letters*, vol. 27, no. 14, pp. 1276-1277, 1991.
[151] W. Zhang, J. Lahteenmaki, and P. Vainikainen, "A practical aspect of over-rooftop multiple-building forward diffraction from a low source," *IEEE transactions on electromagnetic compatibility*, vol. 41, no. 2, pp. 115-119, 1999.

[152] J. Li, J.-F. Wagen, and E. Lachat, "Propagation over rooftop and in the horizontal plane for small and micro-cell coverage predictions," in *1997 IEEE 47th Vehicular Technology Conference. Technology in Motion*, 1997, vol. 2: IEEE, pp. 1123-1127.

[153] "Download Paris Digital Map and Measurements." https://propagationtools.com/wireless/download-paris-digital-map-and-measurements/ (accessed 16/01, 2020).

Appendix I GORL OpenCL Kernels

```
#ifdef _MSC_VER
#  define PACKED_STRUCT(name) \
    __pragma(pack(push, 1)) struct name __pragma(pack(pop))
#  define ALIGNED_STRUCT(name, byteBoundary) __declspec(align(byteBoundary))
struct name
#  define NEW_FLOAT4(x, y, z, w) float4(x, y, z, w)
#  define NEW_FLOAT3(x, y, z) float3(x, y, z)
#  define __localdecl static
#else
#  define PACKED_STRUCT(name) struct __attribute__((packed)) name
#  define ALIGNED_STRUCT(name, byteBoundary) struct
__attribute__((aligned(byteBoundary))) name
#  pragma OPENCL EXTENSION cl_khr_fp64 : disable
#  define NEW_FLOAT4(x, y, z, w) (float4)(x, y, z, w)
#  define NEW_FLOAT3(x, y, z) (float3)(x, y, z)
#  define __localdecl __local
#endif

typedef struct
{
      int SizeOfRay;
      int SizeOfTriangle;
      int SizeOfHitDetails;
      int SizeOfBoundingBox;
      int SizeOfBVHNode;
      int SizeOfGrid;
} DebugInfo;

typedef struct
{
      int NumberOfTriangles;
      int NumberOfRaysToProcess;
      int NumberOfRaysProcessed;
      int __;
} CustomParams;

typedef ALIGNED_STRUCT(__BoundingBox, 4)
{
      float MinX, MinY, MinZ;
      float MaxX, MaxY, MaxZ;
} BoundingBox;

typedef unsigned short NodePtr;
typedef ALIGNED_STRUCT(__BVHNode, 16)
{
      BoundingBox BoundingBox;
      NodePtr BranchLPointer;
      NodePtr BranchRPointer;
      int TriangleIndex;
      int TriangleCount;
```

```
	bool IsLeaf;
} BVHNode;

typedef ALIGNED_STRUCT(__HitDetails, 16)
{
	float3 Position;
	float T;
	float U;
	float V;
	int TriangleIndex;
} HitDetails;

typedef ALIGNED_STRUCT(__Ray, 16)
{
	float3 Origin;
	float3 Direction;
	float StartingDistance;
	float InteractionLoss;
	int Depth;//4
} Ray;

typedef struct __Matrix4x4
{
	float M[16];
} Matrix4x4;

typedef ALIGNED_STRUCT(__Triangle, 16)
{
	float3 A; //16
	float3 B; //16
	float3 C; //16
	int MaterialIndex;
	struct __Matrix4x4 InverseMatrix;
} Triangle;


typedef PACKED_STRUCT(__Grid)
{
	float3 Position;
	float3 VoxelSize;
	int3 Extent;
} Grid;


inline float4 Transform(Matrix4x4 m, float4 v)
{
	return NEW_FLOAT4(
		m.M[0] * v.x + m.M[4] * v.y + m.M[8] * v.z + m.M[12] * v.w,
		m.M[1] * v.x + m.M[5] * v.y + m.M[9] * v.z + m.M[13] * v.w,
		m.M[2] * v.x + m.M[6] * v.y + m.M[10] * v.z + m.M[14] * v.w,
		m.M[3] * v.x + m.M[7] * v.y + m.M[11] * v.z + m.M[15] * v.w);
}

inline Matrix4x4 CreateMatrixFromColumns(float m[16])
{
```

```
	Matrix4x4 mat;
	for (char idx = 0; idx < 16; idx++)
		mat.M[idx] = m[idx];
	return mat;
}

inline Matrix4x4 Invert(Matrix4x4 m)
{
	float inv[16] = {
		m.M[5] * m.M[10] * m.M[15] -
		m.M[5] * m.M[11] * m.M[14] -
		m.M[9] * m.M[6] * m.M[15] +
		m.M[9] * m.M[7] * m.M[14] +
		m.M[13] * m.M[6] * m.M[11] -
		m.M[13] * m.M[7] * m.M[10],

		-m.M[1] * m.M[10] * m.M[15] +
		m.M[1] * m.M[11] * m.M[14] +
		m.M[9] * m.M[2] * m.M[15] -
		m.M[9] * m.M[3] * m.M[14] -
		m.M[13] * m.M[2] * m.M[11] +
		m.M[13] * m.M[3] * m.M[10],

		m.M[1] * m.M[6] * m.M[15] -
		m.M[1] * m.M[7] * m.M[14] -
		m.M[5] * m.M[2] * m.M[15] +
		m.M[5] * m.M[3] * m.M[14] +
		m.M[13] * m.M[2] * m.M[7] -
		m.M[13] * m.M[3] * m.M[6],

		-m.M[1] * m.M[6] * m.M[11] +
		m.M[1] * m.M[7] * m.M[10] +
		m.M[5] * m.M[2] * m.M[11] -
		m.M[5] * m.M[3] * m.M[10] -
		m.M[9] * m.M[2] * m.M[7] +
		m.M[9] * m.M[3] * m.M[6],

		-m.M[4] * m.M[10] * m.M[15] +
		m.M[4] * m.M[11] * m.M[14] +
		m.M[8] * m.M[6] * m.M[15] -
		m.M[8] * m.M[7] * m.M[14] -
		m.M[12] * m.M[6] * m.M[11] +
		m.M[12] * m.M[7] * m.M[10],

		m.M[0] * m.M[10] * m.M[15] -
		m.M[0] * m.M[11] * m.M[14] -
		m.M[8] * m.M[2] * m.M[15] +
		m.M[8] * m.M[3] * m.M[14] +
		m.M[12] * m.M[2] * m.M[11] -
		m.M[12] * m.M[3] * m.M[10],

		-m.M[0] * m.M[6] * m.M[15] +
		m.M[0] * m.M[7] * m.M[14] +
		m.M[4] * m.M[2] * m.M[15] -
		m.M[4] * m.M[3] * m.M[14] -
```

```
            m.M[12] * m.M[2] * m.M[7] +
            m.M[12] * m.M[3] * m.M[6],

            m.M[0] * m.M[6] * m.M[11] -
            m.M[0] * m.M[7] * m.M[10] -
            m.M[4] * m.M[2] * m.M[11] +
            m.M[4] * m.M[3] * m.M[10] +
            m.M[8] * m.M[2] * m.M[7] -
            m.M[8] * m.M[3] * m.M[6],

            m.M[4] * m.M[9] * m.M[15] -
            m.M[4] * m.M[11] * m.M[13] -
            m.M[8] * m.M[5] * m.M[15] +
            m.M[8] * m.M[7] * m.M[13] +
            m.M[12] * m.M[5] * m.M[11] -
            m.M[12] * m.M[7] * m.M[9],

            -m.M[0] * m.M[9] * m.M[15] +
            m.M[0] * m.M[11] * m.M[13] +
            m.M[8] * m.M[1] * m.M[15] -
            m.M[8] * m.M[3] * m.M[13] -
            m.M[12] * m.M[1] * m.M[11] +
            m.M[12] * m.M[3] * m.M[9],

            m.M[0] * m.M[5] * m.M[15] -
            m.M[0] * m.M[7] * m.M[13] -
            m.M[4] * m.M[1] * m.M[15] +
            m.M[4] * m.M[3] * m.M[13] +
            m.M[12] * m.M[1] * m.M[7] -
            m.M[12] * m.M[3] * m.M[5],

            -m.M[0] * m.M[5] * m.M[11] +
            m.M[0] * m.M[7] * m.M[9] +
            m.M[4] * m.M[1] * m.M[11] -
            m.M[4] * m.M[3] * m.M[9] -
            m.M[8] * m.M[1] * m.M[7] +
            m.M[8] * m.M[3] * m.M[5],


            -m.M[4] * m.M[9] * m.M[14] +
            m.M[4] * m.M[10] * m.M[13] +
            m.M[8] * m.M[5] * m.M[14] -
            m.M[8] * m.M[6] * m.M[13] -
            m.M[12] * m.M[5] * m.M[10] +
            m.M[12] * m.M[6] * m.M[9],

            m.M[0] * m.M[9] * m.M[14] -
            m.M[0] * m.M[10] * m.M[13] -
            m.M[8] * m.M[1] * m.M[14] +
            m.M[8] * m.M[2] * m.M[13] +
            m.M[12] * m.M[1] * m.M[10] -
            m.M[12] * m.M[2] * m.M[9],

            -m.M[0] * m.M[5] * m.M[14] +
            m.M[0] * m.M[6] * m.M[13] +
```

```
            m.M[4] * m.M[1] * m.M[14] -
            m.M[4] * m.M[2] * m.M[13] -
            m.M[12] * m.M[1] * m.M[6] +
            m.M[12] * m.M[2] * m.M[5],

            m.M[0] * m.M[5] * m.M[10] -
            m.M[0] * m.M[6] * m.M[9] -
            m.M[4] * m.M[1] * m.M[10] +
            m.M[4] * m.M[2] * m.M[9] +
            m.M[8] * m.M[1] * m.M[6] -
            m.M[8] * m.M[2] * m.M[5] };

      float det = m.M[0] * inv[0] +
            m.M[1] * inv[4] +
            m.M[2] * inv[8] +
            m.M[3] * inv[12];


      //    if (det == 0)
      //    return false;

      det = 1.0f / det;

      for (char i = 0; i < 16; i++)
            inv[i] *= det;

      return CreateMatrixFromColumns(inv);
}

inline bool IsTrueIntersection(float T, float U, float V)
{
      return U >= 0.0f &&
            V >= 0.0f &&
            U + V <= 1.0f
            && (INFINITY != T)
            && T > 1E-6f;
}

inline Matrix4x4 CalculateInverseMatrix(float3 A, float3 B, float3 C)
{
      float3 vectorAB = B - A; //U
      float3 vectorAC = C - A; //V
      float3 normal = cross(vectorAC, vectorAB);
      float cols[16] = {
            vectorAB.x, vectorAB.y, vectorAB.z, 0.0f,
            vectorAC.x, vectorAC.y, vectorAC.z, 0.0f,
            normal.x, normal.y, normal.z, 0.0f,
            A.x, A.y, A.z, 1.0f };

      return Invert(CreateMatrixFromColumns(cols));
}

inline HitDetails Intersect(Triangle tri, Ray r)
{
      float3 vectorAB = tri.B - tri.A;
```

```
	float3 vectorAC = tri.C - tri.A;

	float4 oNP = Transform(tri.InverseMatrix, NEW_FLOAT4(r.Origin.x,
r.Origin.y, r.Origin.z, 1.0f));
	float4 nNP = Transform(tri.InverseMatrix, NEW_FLOAT4(r.Direction.x,
r.Direction.y, r.Direction.z, 0.0f));
	float t = -oNP.z / nNP.z;
	float u = oNP.x + t * nNP.x;
	float v = oNP.y + t * nNP.y;

	HitDetails h;
	h.T = t;
	h.U = u;
	h.V = v;
	h.Position = vectorAB * u + vectorAC * v + tri.A;

	return h;
}
__kernel void DebugKernel(__global DebugInfo* debugInfo)
{
	if (get_global_id(0) == 0)
	{
		debugInfo->SizeOfBoundingBox = sizeof(BoundingBox);
		debugInfo->SizeOfBVHNode = sizeof(BVHNode);
		debugInfo->SizeOfGrid = sizeof(Grid);
		debugInfo->SizeOfHitDetails = sizeof(HitDetails);
		debugInfo->SizeOfRay = sizeof(Ray);
		debugInfo->SizeOfTriangle = sizeof(Triangle);
	};
}

#define MAX(a,b) ((a) > (b) ? a : b)
#define MIN(a,b) ((a) < (b) ? a : b)

inline float CalculateAbsolute(float value)
{
	return value >= 0 ? value : -value;
}

inline float CalculateDistanceByDirection(float3 point1, float3 point2, float3
direction)
{
	if (direction.x != 0)
		return CalculateAbsolute(point1.x - point2.x) / direction.x;
	else if (direction.y != 0)
		return CalculateAbsolute(point1.y - point2.y) / direction.y;
	else
		return CalculateAbsolute(point1.z - point2.z) / direction.z;
}

inline float BoundingBoxIntersect(BoundingBox bb, Ray r)
{
	// r.dir is unit direction vector of ray
	float3 dirFrac = NEW_FLOAT3(1.0f / r.Direction.x, 1.0f / r.Direction.y,
1.0f / r.Direction.z);
```

```
	// lb is the corner of AABB with minimal coordinates - left bottom, rt
is maximal corner
	// r.org is origin of ray
	float t1 = (bb.MinX - r.Origin.x) * dirFrac.x;
	float t2 = (bb.MaxX - r.Origin.x) * dirFrac.x;
	float t3 = (bb.MinY - r.Origin.y) * dirFrac.y;
	float t4 = (bb.MaxY - r.Origin.y) * dirFrac.y;
	float t5 = (bb.MinZ - r.Origin.z) * dirFrac.z;
	float t6 = (bb.MaxZ - r.Origin.z) * dirFrac.z;

	float tmin = max(max(min(t1, t2), min(t3, t4)), min(t5, t6));
	float tmax = min(min(max(t1, t2), max(t3, t4)), max(t5, t6));


	if (tmax < 0 ||// if tmax < 0, ray (line) is intersecting AABB, but the
whole AABB is behind us
		tmin > tmax)// if tmin > tmax, ray doesn't intersect AABB
		return INFINITY;

	return tmin;
}

//Walking collection

typedef PACKED_STRUCT(__StepRange)
{
	float ToNextBoundary;
	float Delta;
	int Step;
} StepRange;

inline int3 GetFirstEntry(Ray ray, Grid grid)
{
	float3 originVector = ray.Origin - grid.Position;
	int3 index;
	index.x = (int)(originVector.x / grid.VoxelSize.x);
	index.y = (int)(originVector.y / grid.VoxelSize.y);
	index.z = (int)(originVector.z / grid.VoxelSize.z);
	return index;
}


inline StepRange GetStepRange(int index, float cellSize, float gridStart,
float rayStart, float projectionRatio)
{
	StepRange stepRange;

	if (projectionRatio < 0)
	{
		stepRange.ToNextBoundary = (gridStart + index * cellSize -
rayStart) / projectionRatio;
		stepRange.Delta = -cellSize / projectionRatio;
		stepRange.Step = -1;
		return stepRange;
```

```
	}

	if (projectionRatio > 0)
	{
		stepRange.ToNextBoundary = (gridStart + (index + 1) * cellSize -
rayStart) / projectionRatio;
		stepRange.Delta = cellSize / projectionRatio;
		stepRange.Step = 1;
		return stepRange;
	}

	stepRange.ToNextBoundary = FLT_MAX;
	stepRange.Delta = FLT_MAX;
	stepRange.Step = 1;
	return stepRange;
}

inline bool IsInside(int3 index, Grid grid)
{
	return 0 <= index.x && index.x < grid.Extent.x
		&& 0 <= index.y && index.y < grid.Extent.y
		&& 0 <= index.z && index.z < grid.Extent.z;
}

inline int GetFlattedIndex(int3 index, int3 extent)
{
	int flattedIndex = index.x + index.y * extent.x + index.z * extent.x *
extent.y;
	return flattedIndex;
}

inline void TrianglesIntersection(HitDetails* bestHit, Ray ray, int
triangleCount, __global Triangle* triangles, int firstTriangleIndex)
{
	for (int triangleIndex = 0; triangleIndex < triangleCount;
triangleIndex++)
	{
		Triangle tri = triangles[triangleIndex + firstTriangleIndex];
		HitDetails currHit = Intersect(tri, ray);

		if (IsTrueIntersection(currHit.T, currHit.U, currHit.V) &&
			currHit.T < bestHit->T)
		{
			*bestHit = currHit;
			bestHit->TriangleIndex = triangleIndex +
firstTriangleIndex;
		}
	}
}

inline HitDetails VTMIntersection(__global int* voxelPointers,
	__global int* triangleIndexPointers, Ray currentRay, __global Triangle*
triangles,	__global const Grid* grid)

{
```

```
	const Grid gr = *grid;
	int3 index = GetFirstEntry(currentRay, gr);
	StepRange stepRangeX = GetStepRange(index.x, gr.VoxelSize.x,
gr.Position.x, currentRay.Origin.x, currentRay.Direction.x);
	StepRange stepRangeY = GetStepRange(index.y, gr.VoxelSize.y,
gr.Position.y, currentRay.Origin.y, currentRay.Direction.y);
	StepRange stepRangeZ = GetStepRange(index.z, gr.VoxelSize.z,
gr.Position.z, currentRay.Origin.z, currentRay.Direction.z);
	HitDetails bestHit;

	while (IsInside(index, gr))
	{
		int flattenIndex = GetFlattedIndex(index, gr.Extent);
		int triangleIndexPtrOffset = voxelPointers[flattenIndex];
		__global int* triangleIndices = triangleIndexPointers +
triangleIndexPtrOffset;
		int triangleCount = triangleIndices[0];

		bestHit.T = INFINITY;

		for (int i = 0; i < triangleCount; i++)
		{
			const Triangle triangle = triangles[triangleIndices[i]];
			HitDetails currHit = Intersect(triangle, currentRay);

			if (IsTrueIntersection(currHit.T, currHit.U, currHit.V) &&
				currHit.T < bestHit.T)
			{
				bestHit = currHit;
				bestHit.TriangleIndex = triangleIndices[i];
			}
		}

		if (bestHit.T != INFINITY)
		{
			return bestHit;
		}

		if (stepRangeX.ToNextBoundary < stepRangeY.ToNextBoundary)
		{
			if (stepRangeX.ToNextBoundary < stepRangeZ.ToNextBoundary)
			{
				index.x += stepRangeX.Step;
				stepRangeX.ToNextBoundary += stepRangeX.Delta;
			}
			else
			{
				index.z += stepRangeZ.Step;
				stepRangeZ.ToNextBoundary += stepRangeZ.Delta;
			}
		}
		else
		{
			if (stepRangeY.ToNextBoundary < stepRangeZ.ToNextBoundary)
			{
```

```
                        index.y += stepRangeY.Step;
                        stepRangeY.ToNextBoundary += stepRangeY.Delta;
                    }
                    else
                    {
                        index.z += stepRangeZ.Step;
                        stepRangeZ.ToNextBoundary += stepRangeZ.Delta;
                    }
                }
            }

            return bestHit;
}

inline HitDetails BVHIntersection(__global BVHNode* flatBVHTree, Ray ray,
__global Triangle* triangles)
{
            // Allocate traversal stack from thread-local memory,
            // and push NULL to indicate that there are no postponed nodes.
            NodePtr stack[16];
            NodePtr* stackPtr = stack;
            *stackPtr++ = 0; // push

            HitDetails bestHit;
            bestHit.T = INFINITY;

            NodePtr node = 0;
            do
            {
                BVHNode currentNode = flatBVHTree[node];

                bool overlap = BoundingBoxIntersect(currentNode.BoundingBox, ray)
< bestHit.T;

                if (overlap && currentNode.IsLeaf)
                    TrianglesIntersection(&bestHit, ray,
currentNode.TriangleCount, triangles, currentNode.TriangleIndex);

                bool traverseL = (currentNode.BranchLPointer != 0) && overlap;
                bool traverseR = (currentNode.BranchRPointer != 0) && overlap;

                float leftDistance = INFINITY;
                float rightDistance = INFINITY;
                if (traverseL)
                    leftDistance =
BoundingBoxIntersect(flatBVHTree[currentNode.BranchLPointer].BoundingBox,
ray);

                if (traverseR)
                    rightDistance =
BoundingBoxIntersect(flatBVHTree[currentNode.BranchRPointer].BoundingBox,
ray);

                if (leftDistance >= bestHit.T && rightDistance >= bestHit.T)
                    node = *--stackPtr; // pop
```

```
            else
            {
                  NodePtr nearestPtr = leftDistance < rightDistance ?
currentNode.BranchLPointer : currentNode.BranchRPointer;
                  NodePtr farthestPtr = leftDistance < rightDistance ?
currentNode.BranchRPointer : currentNode.BranchLPointer;

                  node = nearestPtr;
                  if (leftDistance < bestHit.T && rightDistance < bestHit.T)
                        *stackPtr++ = farthestPtr; // push
            }
      } while (node != 0);

      return bestHit;
}


__kernel void CalculateInverseMatrices(
      __global BVHNode* flatBVHTree,
      __constant CustomParams* customParams,
      __global const Grid* grid,
      __global const Ray* rays,
      __global Triangle* triangles,
      __global HitDetails* hits,
      __global int* voxelValues)
{
      int threadIndex = get_global_id(0);
      if (threadIndex >= customParams->NumberOfTriangles)
            return;

      __global Triangle* tri = triangles + threadIndex;
      tri->InverseMatrix = CalculateInverseMatrix(tri->A, tri->B, tri->C);
}


__kernel void VTMIntersectionKernel(
      __global int* voxelPointers,
      __global int* triangleIndexPointers,
      __constant CustomParams* customParams,
      __global const Grid* grid,
      __global const Ray* rays,
      __global Triangle* triangles,
      __global HitDetails* hits,
      __global int* voxelValues)
{
      int threadIndex = get_global_id(0);
      if (threadIndex >= customParams->NumberOfRaysToProcess)
            return;
      Ray currentRay = rays[threadIndex];
      hits[threadIndex] = VTMIntersection(voxelPointers,
triangleIndexPointers, currentRay, triangles, grid);
}

__kernel void BVHIntersectionKernel(
      __global BVHNode* flatBVHTree,
```

```
	__constant CustomParams* customParams,
	__global const Grid* grid,
	__global const Ray* rays,
	__global Triangle* triangles,
	__global HitDetails* hits,
	__global int* voxelValues)
{
	int threadIndex = get_global_id(0);
	if (threadIndex >= customParams->NumberOfRaysToProcess)
		return;
	Ray currentRay = rays[threadIndex];
	hits[threadIndex] = BVHIntersection(flatBVHTree, currentRay, triangles);
}

inline float GetVectorLength(float3 vector)
{
	return sqrt(vector.x * vector.x + vector.y * vector.y + vector.z *
vector.z);
}

inline float3 GetVoxelCenter(Grid grid, int3 index)
{
	float3 center;
	center.x = grid.Position.x + (index.x + 0.5f) * grid.VoxelSize.x;
	center.y = grid.Position.y + (index.y + 0.5f) * grid.VoxelSize.y;
	center.z = grid.Position.z + (index.z + 0.5f) * grid.VoxelSize.z;
	return center;
}

inline int CalculatePathlossByDistance(float distance)
{
	const float pathlossMultiplier = 75.4f;
	return (int)(1000 * (33 * log10(pathlossMultiplier * distance) -
52.9f));
}


__kernel void CollectResultByWalking(
	__global BVHNode* flatBVHTree,
	__constant CustomParams* customParams,
	__global const Grid* grid,
	__global const Ray* rays,
	__global Triangle* triangles,
	__global HitDetails* hits,
	__global int* voxelValues
)
{
	int threadIndex = get_global_id(0);
	if (threadIndex >= customParams->NumberOfRaysToProcess)
		return;
	Grid gr = *grid;
	HitDetails hit = hits[threadIndex];
	Ray currentRay = rays[threadIndex];

	float totalDistance = IsTrueIntersection(hit.T, hit.U, hit.V)
```

```
            ? GetVectorLength(hit.Position - currentRay.Origin)
            : FLT_MAX;

      float travelDistance = 0;

      int3 index = GetFirstEntry(currentRay, gr);
      StepRange stepRangeX = GetStepRange(index.x, gr.VoxelSize.x,
gr.Position.x, currentRay.Origin.x, currentRay.Direction.x);
      StepRange stepRangeY = GetStepRange(index.y, gr.VoxelSize.y,
gr.Position.y, currentRay.Origin.y, currentRay.Direction.y);
      StepRange stepRangeZ = GetStepRange(index.z, gr.VoxelSize.z,
gr.Position.z, currentRay.Origin.z, currentRay.Direction.z);

      while (travelDistance < totalDistance && IsInside(index, gr))
      {
            float3 travelVector = GetVoxelCenter(gr, index) -
currentRay.Origin;
            float distance = GetVectorLength(travelVector) +
currentRay.StartingDistance;
            int pathloss = CalculatePathlossByDistance(distance) +
currentRay.InteractionLoss;
            int flattenIndex = GetFlattedIndex(index, gr.Extent);
            atomic_min(voxelValues + flattenIndex, pathloss);

            if (stepRangeX.ToNextBoundary < stepRangeY.ToNextBoundary)
            {
                  if (stepRangeX.ToNextBoundary < stepRangeZ.ToNextBoundary)
                  {
                        index.x += stepRangeX.Step;
                        travelDistance = stepRangeX.ToNextBoundary;
                        stepRangeX.ToNextBoundary += stepRangeX.Delta;
                  }
                  else
                  {
                        index.z += stepRangeZ.Step;
                        travelDistance = stepRangeZ.ToNextBoundary;
                        stepRangeZ.ToNextBoundary += stepRangeZ.Delta;
                  }
            }
            else
            {
                  if (stepRangeY.ToNextBoundary < stepRangeZ.ToNextBoundary)
                  {
                        index.y += stepRangeY.Step;
                        travelDistance = stepRangeY.ToNextBoundary;
                        stepRangeY.ToNextBoundary += stepRangeY.Delta;
                  }
                  else
                  {
                        index.z += stepRangeZ.Step;
                        travelDistance = stepRangeZ.ToNextBoundary;
                        stepRangeZ.ToNextBoundary += stepRangeZ.Delta;
                  }
            }
      }
```

```
}

inline void CollectRay(HitDetails hit, Ray currentRay, Grid gr, __global int*
voxelValues)
{
	float totalDistance = IsTrueIntersection(hit.T, hit.U, hit.V)
		? GetVectorLength(hit.Position - currentRay.Origin)
		: FLT_MAX;

	float travelDistance = 0;

	int3 index = GetFirstEntry(currentRay, gr);
	StepRange stepRangeX = GetStepRange(index.x, gr.VoxelSize.x,
gr.Position.x, currentRay.Origin.x, currentRay.Direction.x);
	StepRange stepRangeY = GetStepRange(index.y, gr.VoxelSize.y,
gr.Position.y, currentRay.Origin.y, currentRay.Direction.y);
	StepRange stepRangeZ = GetStepRange(index.z, gr.VoxelSize.z,
gr.Position.z, currentRay.Origin.z, currentRay.Direction.z);

	while (travelDistance < totalDistance && IsInside(index, gr))
	{
		float3 travelVector = GetVoxelCenter(gr, index) -
currentRay.Origin;
		float distance = GetVectorLength(travelVector) +
currentRay.StartingDistance;
		int pathloss = CalculatePathlossByDistance(distance) +
currentRay.InteractionLoss;
		int flattenIndex = GetFlattedIndex(index, gr.Extent);
		atomic_min(voxelValues + flattenIndex, pathloss);

		if (stepRangeX.ToNextBoundary < stepRangeY.ToNextBoundary)
		{
			if (stepRangeX.ToNextBoundary < stepRangeZ.ToNextBoundary)
			{
				index.x += stepRangeX.Step;
				travelDistance = stepRangeX.ToNextBoundary;
				stepRangeX.ToNextBoundary += stepRangeX.Delta;
			}
			else
			{
				index.z += stepRangeZ.Step;
				travelDistance = stepRangeZ.ToNextBoundary;
				stepRangeZ.ToNextBoundary += stepRangeZ.Delta;
			}
		}
		else
		{
			if (stepRangeY.ToNextBoundary < stepRangeZ.ToNextBoundary)
			{
				index.y += stepRangeY.Step;
				travelDistance = stepRangeY.ToNextBoundary;
				stepRangeY.ToNextBoundary += stepRangeY.Delta;
			}
			else
			{
```

```
                        index.z += stepRangeZ.Step;
                        travelDistance = stepRangeZ.ToNextBoundary;
                        stepRangeZ.ToNextBoundary += stepRangeZ.Delta;
                    }
                }
        }
}

inline float DotProduct(float3 a, float3 b)
{
        return a.x*b.x + a.y*b.y + a.z*b.z;
}

inline float3 Normalize(float3 v)
{
        float length = sqrt(v.x*v.x + v.y*v.y + v.z*v.z);
        return v / length;
}

inline Ray GenerateNewRay(Ray currentRay, HitDetails hit, short newRayIndex,
__global const Triangle* triangles)
{
        float lastDistance = GetVectorLength(hit.Position-currentRay.Origin);

        Ray newRay;
        newRay.Depth = currentRay.Depth + 1;
        newRay.Origin = hit.Position;
        newRay.StartingDistance = currentRay.StartingDistance + lastDistance;

        if (newRayIndex == 0)
        {
                newRay.Direction = currentRay.Direction;
                newRay.InteractionLoss = currentRay.InteractionLoss +
9100/*triangles[hit.TriangleIndex].MaterialIndex*/;//TODO:Need material
        }
        else
        {
                Triangle tri = triangles[hit.TriangleIndex];
                float3 vectorAB = tri.A - tri.B;
                float3 vectorAC = tri.A - tri.C;
                float3 normal = cross(vectorAC, vectorAB);
                normal = Normalize(normal);

                float3 vectorFrom = hit.Position - currentRay.Origin;
                float3 vectorFromProjectedOnNormal = normal * DotProduct(normal,
vectorFrom);
                float3 reflectionDirection = vectorFrom -
vectorFromProjectedOnNormal * 2;

                newRay.InteractionLoss = currentRay.InteractionLoss + 7300;
                newRay.Direction = Normalize(reflectionDirection);
        }
        return newRay;
}
```

```
__kernel void RayIntersectionAndCollection(
	__global BVHNode* flatBVHTree,
	__global CustomParams* customParams,
	__global const Grid* grid,
	__global const Ray* rays,
	__global Triangle* triangles,
	__global HitDetails* hits,
	__global int* voxelValues)
{
	int threadIndex = get_global_id(0);
	if (threadIndex >= customParams->NumberOfRaysToProcess)
		return;

	Grid gr = *grid;
	Ray currentRay = rays[threadIndex];

	HitDetails hitStack[5];
	short depthCountStack[5];
	Ray rayStack[5];

	HitDetails* hitStackPtr = hitStack;
	short* depthCountStactPtr = depthCountStack;
	Ray* rayStackPtr = rayStack;

	HitDetails hit = BVHIntersection(flatBVHTree, currentRay, triangles);
	CollectRay(hit, currentRay, gr, voxelValues);
	short currentDepthCount = 0;

	if (hit.T == INFINITY)
	{
		return;
	}

	*hitStackPtr = hit;
	*depthCountStactPtr = currentDepthCount;
	*rayStackPtr = currentRay;

	do
	{
		if (currentRay.Depth >= 4 || hit.T == INFINITY ||
currentDepthCount >= 2)
		{
			hit = *--hitStackPtr;
			currentDepthCount = *--depthCountStactPtr;
			currentRay = *--rayStackPtr;
		}

		if (currentDepthCount < 2)
		{
			currentRay = GenerateNewRay(currentRay, hit,
currentDepthCount, triangles);
			hit = BVHIntersection(flatBVHTree, currentRay, triangles);
			atomic_inc(&customParams->NumberOfRaysProcessed);
			CollectRay(hit, currentRay, gr, voxelValues);
			(*depthCountStactPtr)++;
```

```
                    currentDepthCount = 0;

                    *++hitStackPtr = hit;
                    *++depthCountStactPtr = currentDepthCount;
                    *++rayStackPtr = currentRay;
                }

        } while (hitStackPtr != hitStack || currentDepthCount < 2);
}
```

Appendix II TLARS OpenCL Kernels

```
#include "CommonTypes.cl.h"
#include "RayCollectionKernel.cl.h"
#include "GridTraversal.cl.h"

#ifdef OPENCLCOLLECTORKERNEL_ENABLE_STATS
#pragma OPENCL EXTENSION cl_khr_global_int32_base_atomics : enable
#endif

#if !defined(OPENCL_COMPILER) && !defined(__NVCC__)

namespace Ranplan::Maxwell::HPC
{
#endif
    DEVICEFUNC inline float CalculatePathlossByDistance(float distance, float
spaceLossConstant, float offset, float exponentialFactor, float linearFactor)
    {
        return exponentialFactor * 10.0f * log10(distance) + linearFactor *
distance + offset + spaceLossConstant;
    }

    DEVICEFUNC inline float GetVectorLength(float4 vector)
    {
        return sqrt(vector.x * vector.x + vector.y * vector.y + vector.z *
vector.z);
    }

    DEVICEFUNC inline float4 GetVoxelCenter(Grid grid, int3 index)
    {
        float4 center;
        center.x = grid.Position.x + (index.x + 0.5f) * grid.VoxelSize.x;
        center.y = grid.Position.y + (index.y + 0.5f) * grid.VoxelSize.y;
        center.z = grid.Position.z + (index.z + 0.5f) * grid.VoxelSize.z;
        return center;
    }

    DEVICEFUNC inline float Power(float x, float y)
    {
        //Intel OCL compiler bug workaround
        return exp(log(x) * y);
    }

    DEVICEFUNC inline float GetLinearPower(float pathloss)
    {
        //Assume 0dBm transmit power
        return Power(10.0f, -pathloss / 10.0f);
    }

    DEVICEFUNC inline float AddPathloss(float pathlossA, float pathlossB)
    {
        float newPower = GetLinearPower(pathlossA) +
GetLinearPower(pathlossB);
```

```
        return 0 - 10.0f * log10(newPower);
    }

    DEVICEFUNC inline int GetGlobalIndex(int4 globalExtent, int3 globalIndex)
    {
        return globalIndex.z * globalExtent.x * globalExtent.y + globalIndex.y
* globalExtent.x +
            globalIndex.x;
    }

    DEVICEFUNC inline void atomicAdd_g_f(__global float *addr, float val)
    {
        union {
            unsigned int u32;
            float        f32;
        } next, expected, current;
        current.f32 = *addr;
        do {
            expected.f32 = current.f32;
            next.f32 = expected.f32 + val;
            current.u32 = atomic_cmpxchg((__global unsigned int *)addr,
                expected.u32, next.u32);
        } while (current.u32 != expected.u32);
    }


    __kernel void CollectionKernel(
        CollectionParams customParams,
        Grid grid,
        __global float* pathlosses,
        __global unsigned int* flags,
        __global const Ray* rays,
        __global const RayPayload* payloads,
        __global const HitDetails* hits)
    {
        size_t threadIndex = get_global_id(0);

        if (threadIndex >= customParams.NumberOfRaysToProcess)
            return;
        Ray currentRay = rays[threadIndex];
        GridTraversalEnumerable walker;
        float firstHitDistance, _;
        if (!CreateGridTraversalEnumerable(grid, currentRay, &walker,
&firstHitDistance, &_))
            return;

        RayPayload payload = payloads[threadIndex];
        HitDetails hit = hits[threadIndex];

        if (firstHitDistance > hit.T)
            return; //The nearest point of this block is beyond the hit-on-
triangle point.

        do
        {
```

```
            // WARNING: It is necessary to test if the step is within the grid
before processing triangles.
            // Due to floating precision error the first step could be outside
of the grid.

            if (IsPartOfGrid(grid, walker.CurrentIndex))
            {
                int globalIndex = GetGlobalIndex(grid.Extent,
walker.CurrentIndex);
                unsigned int oldFlag = atomic_or(flags + globalIndex,
payload.BatchFlag);
                if ((oldFlag & payload.BatchFlag) == 0)
                {
                    float4 travelVector = GetVoxelCenter(grid,
walker.CurrentIndex) - payload.
                        SourceLocation;
                    float distance = GetVectorLength(travelVector) + payload.
                        PreviousDistanceUntilLastDiffraction;
                    float pathloss = CalculatePathlossByDistance(distance,

customParams.SpaceLossConstant,

customParams.SpaceLossOffset,

customParams.SpaceLossExponentialFactor,

customParams.SpaceLossLinearFactor)
                        + payload.PreviousInteractionLoss;
                    float power = powr(10.0f, (165.0f - pathloss) / 10.0f);
                    atomicAdd_g_f(pathlosses + globalIndex, power);
                }
            }

            //Test if we reached the last voxel
            float travelDistance = GetTravelDistance(&walker);
            if (travelDistance > hit.T)
                break;

            MoveToNext(grid, currentRay, &walker);

        } while (IsPartOfGrid(grid, walker.CurrentIndex));
    } //Kernel


    __kernel void MaxCollectionKernel(
        CollectionParams customParams,
        Grid grid,
        __global int* pathlosses,
        __global const Ray* rays,
        __global const RayPayload* payloads,
        __global const HitDetails* hits)
    {
        size_t threadIndex = get_global_id(0);

        if (threadIndex >= customParams.NumberOfRaysToProcess)
```

```
            return;
        Ray currentRay = rays[threadIndex];
        GridTraversalEnumerable walker;
        float firstHitDistance, _;
        if (!CreateGridTraversalEnumerable(grid, currentRay, &walker,
&firstHitDistance, &_))
            return;

        RayPayload payload = payloads[threadIndex];
        HitDetails hit = hits[threadIndex];

        if (firstHitDistance > hit.T)
            return; //The nearest point of this block is beyond the hit-on-
triangle point.

        do
        {
            // WARNING: It is necessary to test if the step is within the grid
before processing triangles.
            // Due to floating precision error the first step could be outside
of the grid.

            if (IsPartOfGrid(grid, walker.CurrentIndex))
            {
                int globalIndex = GetGlobalIndex(grid.Extent,
walker.CurrentIndex);

                float4 travelVector = GetVoxelCenter(grid,
walker.CurrentIndex) - payload.
                    SourceLocation;
                float distance = GetVectorLength(travelVector) + payload.
                    PreviousDistanceUntilLastDiffraction;
                float pathloss = CalculatePathlossByDistance(distance,

customParams.SpaceLossConstant,

customParams.SpaceLossOffset,

customParams.SpaceLossExponentialFactor,

customParams.SpaceLossLinearFactor)
                    + payload.PreviousInteractionLoss;
                int pathlossInt = ToIntPathloss(pathloss);

                atomic_min(pathlosses + globalIndex, pathlossInt);
            }

            //Test if we reached the last voxel
            float travelDistance = GetTravelDistance(&walker);
            if (travelDistance > hit.T)
                break;

            MoveToNext(grid, currentRay, &walker);

        } while (IsPartOfGrid(grid, walker.CurrentIndex));
```

```
    } //Kernel

__kernel void MultipathCountingKernel(
    CollectionParams customParams,
    Grid grid,
    __global const char* collectionFlags,
    __global unsigned int* count,
    __global const Ray* rays,
    __global const RayPayload* payloads,
    __global const HitDetails* hits)
{
    size_t threadIndex = get_global_id(0);

    if (threadIndex >= customParams.NumberOfRaysToProcess)
        return;
    Ray currentRay = rays[threadIndex];
    GridTraversalEnumerable walker;
    float firstHitDistance, _;
    if (!CreateGridTraversalEnumerable(grid, currentRay, &walker,
&firstHitDistance, &_))
        return;

    HitDetails hit = hits[threadIndex];

    if (firstHitDistance > hit.T)
        return; //The nearest point of this block is beyond the hit-on-
triangle point.

    do
    {
        // WARNING: It is necessary to test if the step is within the grid
before processing triangles.
        // Due to floating precision error the first step could be outside of
the grid.

        if (IsPartOfGrid(grid, walker.CurrentIndex))
        {
            int globalIndex = GetGlobalIndex(grid.Extent,
walker.CurrentIndex);
            if(collectionFlags[globalIndex])
            {
                atomic_inc(count);
            }
        }

        //Test if we reached the last voxel
        float travelDistance = GetTravelDistance(&walker);
        if (travelDistance > hit.T)
            break;

        MoveToNext(grid, currentRay, &walker);

    } while (IsPartOfGrid(grid, walker.CurrentIndex));
} //Kernel
```

```
__kernel void MultipathIDCollectionKernel(
    CollectionParams customParams,
    Grid grid,
    __global const char* collectionFlags,
    __global unsigned int* offset,
    __global unsigned int* collectedInitialIndices,
    __global const Ray* rays,
    __global const RayPayload* payloads,
    __global const HitDetails* hits)
{
    size_t threadIndex = get_global_id(0);

    if (threadIndex >= customParams.NumberOfRaysToProcess)
        return;
    Ray currentRay = rays[threadIndex];
    GridTraversalEnumerable walker;
    float firstHitDistance, _;
    if (!CreateGridTraversalEnumerable(grid, currentRay, &walker,
&firstHitDistance, &_))
        return;

    RayPayload payload = payloads[threadIndex];
    HitDetails hit = hits[threadIndex];

    if (firstHitDistance > hit.T)
        return; //The nearest point of this block is beyond the hit-on-
triangle point.

    do
    {
        // WARNING: It is necessary to test if the step is within the grid
before processing triangles.
        // Due to floating precision error the first step could be outside of
the grid.

        if (IsPartOfGrid(grid, walker.CurrentIndex))
        {
            int globalIndex = GetGlobalIndex(grid.Extent,
walker.CurrentIndex);
            if (collectionFlags[globalIndex])
            {
                unsigned int old = atomic_inc(offset);
                collectedInitialIndices[old] =
payload.Identifier.InitialRayIndex;
            }
        }

        //Test if we reached the last voxel
        float travelDistance = GetTravelDistance(&walker);
        if (travelDistance > hit.T)
            break;

        MoveToNext(grid, currentRay, &walker);

    } while (IsPartOfGrid(grid, walker.CurrentIndex));
```

```
} //Kernel

__kernel void RayIndicesCollectionKernel(
    CollectionParams customParams,
    Grid grid,
    __global const int* collectionPointOffsets,
    __global const unsigned int* batchCount,
    __global int* collectedRayIndices,
    __global const Ray* rays,
    __global const RayPayload* payloads,
    __global const HitDetails* hits)
{
    size_t threadIndex = get_global_id(0);

    if (threadIndex >= customParams.NumberOfRaysToProcess)
        return;
    Ray currentRay = rays[threadIndex];
    GridTraversalEnumerable walker;
    float firstHitDistance, _;
    if (!CreateGridTraversalEnumerable(grid, currentRay, &walker,
&firstHitDistance, &_))
        return;

    RayPayload payload = payloads[threadIndex];
    HitDetails hit = hits[threadIndex];

    if (firstHitDistance > hit.T)
        return; //The nearest point of this block is beyond the hit-on-
triangle point.

    do
    {
        // WARNING: It is necessary to test if the step is within the grid
before processing triangles.
        // Due to floating precision error the first step could be outside of
the grid.

        if (IsPartOfGrid(grid, walker.CurrentIndex))
        {
            int globalIndex = GetGlobalIndex(grid.Extent,
walker.CurrentIndex);
            if (collectionPointOffsets[globalIndex] >= 0)
            {
                unsigned int idOffset = (unsigned
int)(collectionPointOffsets[globalIndex] * *batchCount + payload.BatchFlag);
                collectedRayIndices[idOffset] = (int)threadIndex;
            }
        }

        //Test if we reached the last voxel
        float travelDistance = GetTravelDistance(&walker);
        if (travelDistance > hit.T)
            break;

        MoveToNext(grid, currentRay, &walker);
```

```
    } while (IsPartOfGrid(grid, walker.CurrentIndex));
} //Kernel

#if !defined(OPENCL_COMPILER) && !defined(__NVCC__)
}
#endif
```