NIST Quasi-Deterministic (Q-D) Channel Realization Software Documentation

by

Anuraag Bodi, Camillo Gentile, Jiayi Zhang, Tanguy Ropitault and Neeraj Varshney

WIRELESS NETWORKS DIVISION, COMMUNICATION TECHNOLOGY LAB

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY,

GAITHERSBURG, MD, USA

AUGUST, 2019

Contents

List of Figures

Abbreviations

5G	Fifth Generation
AMF	Additive Manufacturing File format
AoA	Angle of Arrival
AoD	Angle of Departure
DoA	Direction of Arrival
DoD	Direction of Departure
GIS	Geographic Information System
LOS	Line Of Sight
NIST	National Institute of Standards and Technology
Q-D	Quasi-Deterministic
STL	STereoLithography
Rx	Receiver
Tx	Transmitter
XML	eXtensible Markup Language

Notation

τ	delay
d	path length
c	velocity of light
λ_0	wavelength
f_0	frequency
Δf	change in frequency due to Doppler effect
$\Delta\phi$	phase shift due to Doppler effect
Δv	relative velocity
K factor	the ratio of specular power to the sum of diffuse power
γ	rate of decrease of power of diffuse components with respect to specular components
λ	delay spread parameter
Θ	Angular spread parameter
θ	AoA/AoD elevation angle
ϕ	AoA/AoD azimuth angle

NIST Q-D Channel Realization Software Documentation

1 Introduction

A wireless channel propagation model can be expressed as a collection of rays each representing a separate electromagnetic wave front propagating through a physical medium between two communicating devices or nodes. Stochastic propagation models, that describe the properties of the rays, are generally being used to characterize the wireless channel for both outdoor as well as indoor environments [1]. As the wireless communication protocols become increasingly complex, these stochastic models are now insufficient for modeling channels, in particular for the IEEE 802.11ay standard for next-generation 60-GHz Wireless-Fidelity (Wi-Fi) systems [2]. With the advent of these standards, channel models that provide the space, time, and phase characteristics of all rays deterministically has become necessary for modeling these complex systems.

Geometrical raytracing is one of the possible techniques to capture the deterministic components of all the rays. However, few physical phenomena like rough surface scattering cannot be modeled in deterministic manner. In this context, the National Institute of Standards and Technology (NIST) has adopted the Quasi-Deterministic (Q-D) channel model proposed by the IEEE 802.11ay task group [3] and extracted pa-

rameters based on rigorous channel measurements [4] to stochastically model the rough surface scattering. Geometrical raytracing can be implemented by extracting the geometrical features of an environment and storing it in Computer Aided Design (CAD) or Geographic Information System (GIS) file formats. NIST Q-D channel realization software has chosen CAD files to extract geometrical features from a given environment since they can be generated easily for both indoor and outdoor environments.

The NIST Q-D channel realization software developed by the NIST has two major engines a deterministic engine and a stochastic engine to compute the rays between communicating nodes. Each of the rays is characterized by the path loss, the phase, the angle-of-arrival (at the receiver (Rx)), and the angle-of-departure (from the transmitter (Tx)). The deterministic engine generates deterministic rays, also referred to as specular rays[1]. Using the deterministic rays as the basis, stochastic rays are generated from them using a statistical distribution function based on the NIST rough surface scattering model.

2 Overview of NIST Q-D channel realization software

NIST Q-D channel realization software is a raytracing software developed in Matlab 2017b, which computes multipath components between communicating nodes. A sample output of NIST Q-D channel realization software for two stationary (or fixed) nodes simulated until second order reflection is shown in Fig. 1. However, it is worth noting that the NIST Q-D channel realization software not only supports stationary nodes but also supports mobile nodes.

[1]Specular rays

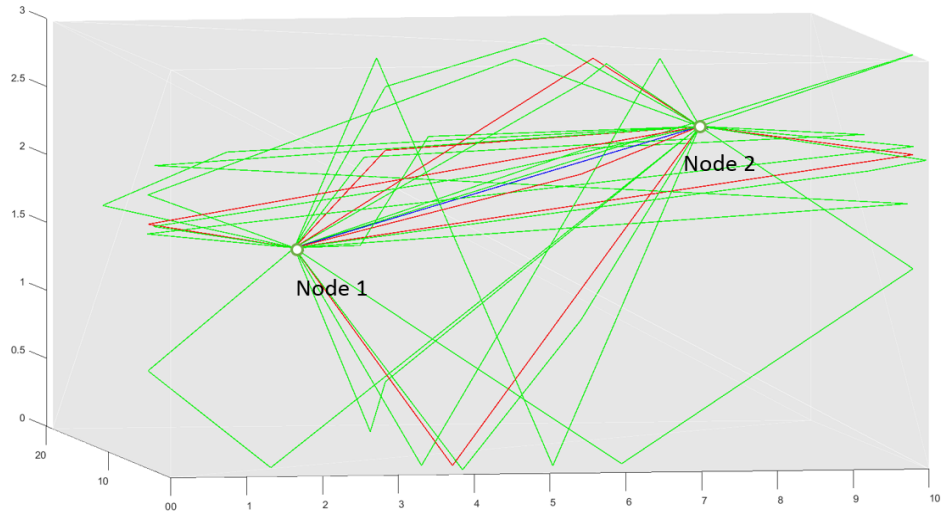

Figure 1: Scenario with two nodes simulated up to second-order reflections. Blue line is the line of sight (LOS), the red lines are first order reflections, and the green lines are second order reflections.

All the features of NIST Q-D channel realization software are described below.

1. Generates specular rays which are dependent on the CAD model of the environment under consideration;

2. Generates nodes either randomly or based on a user-specific placement;

3. Accounts for mobility of the nodes in the environment using a random waypoint model;

4. Generates diffuse rays;

5. Can deal with multiple nodes in the environment.

NIST Q-D channel realization software generates rays based on the node locations and node velocities which can be generated either randomly or based on a user-specific placement. For example, a scenario with three nodes has a total of six possible pair combinations for a fully connected network. These combinations are shown in Table

I and the simulated scenario is shown in Fig. 2 where node locations are generated randomly.

Rx / Tx	Node 1	Node 2	Node 3
Node 1	N/A	Tx1 Rx2	Tx1 Rx3
Node 2	Tx2 Rx1	N/A	Tx2 Rx3
Node 3	Tx3 Rx1	Tx3 Rx2	N/A

Table I: Different combinations of nodes as Transmitter (Tx) and Receiver (Rx).

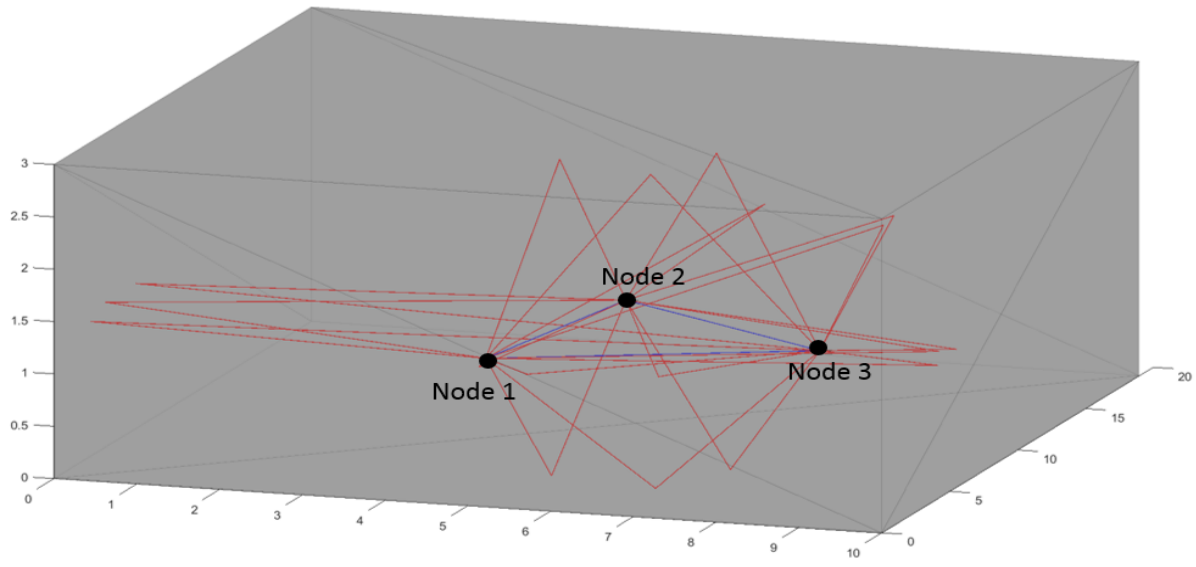

Figure 2: Scenario simulated for multiple stationary nodes which includes LOS (in blue) and first-order reflections (in red).

2.1 Mobility Model

Mobility model is defined for a fixed period of time and the number of divisions that make up this time period. Two modes of mobility are employed in the software linear

mobility and custom mobility.

Linear mobility In linear mobility, the velocities are given by the user or can also be generated randomly. The velocities are given in vectorial form, i.e., (x, y, z) form. Based on the time period and the number of time divisions provided by the user, the node position in each time step[2] is calculated using finite difference method. The new location or position using finite difference method is given as

$$\text{newLocation} = \text{oldLocation} + (\text{velocity} \times \text{timeStep}).$$

For the scenario when the node hits an object/wall, then the software automatically reverses the velocity, i.e., simply multiplies the velocity by -1.

Custom mobility In custom mobility, the node positions of each time step are given by the user. In addition, the value of time step is also provided by the user. Using finite difference method, software calculates the velocity vector at every time step as follows.

$$\text{velocity} = \frac{\text{oldLocation} - \text{newLocation}}{\text{timeStep}}.$$

2.2 Ray Properties

Geometrical properties of a single ray between Tx and Rx are characterized by

- Path length, which is defined as the length of the propagation path of the ray between Tx and Rx.
- The direction of arrival and the direction of departure in the respective coordinate

[2]Time step is defined as the ratio of the time period and the number of divisions.

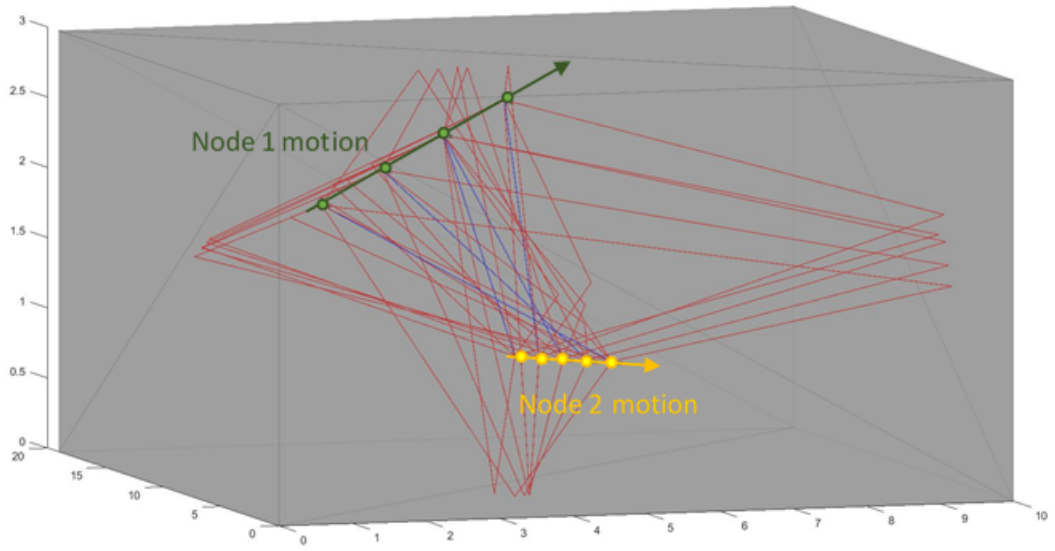

Figure 3: Scenario simulated for mobile nodes which includes LOS (in blue) and first order reflections (in red).

systems of Rx and Tx.

These are the primary attributes of a ray and are shown in the Fig. 4. The directional vector of the ray where the ray starts from Tx is the direction of departure. The angle of this directional vector in the spherical coordinate system is defined as the angle of departure. The negative of directional vector of the ray where the ray ends is the direction of arrival. The angle of the direction of arrival in the spherical coordinate system is defined as the angle of departure. The direction of departure and arrival vectors are considered to be emanating from Tx and Rx, respectively. The angles of departure and arrival has two components azimuth and elevation. The angle of departure in the azimuth plane is calculated by projecting the direction of departure vector onto x-y plane and computing the angle between this projected vector and x axis. The angle of departure in elevation plane is obtained by computing the angle between the direction of departure and the positive z axis. These angles are shown in

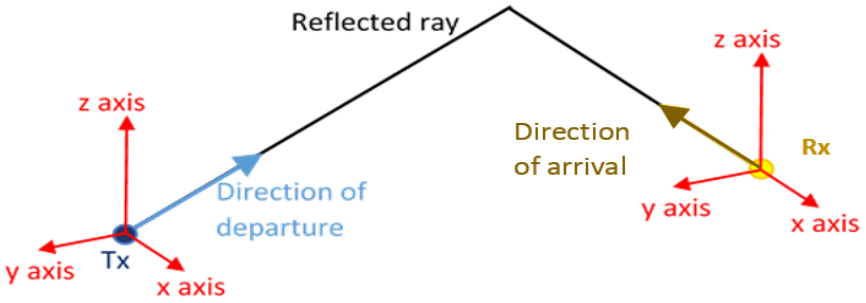

Figure 4: Reflected ray and its direction of arrival and departure vectors.

the Fig. 5 for better clarity. Similarly, the angles of arrival in azimuth and elevation planes can be computed.

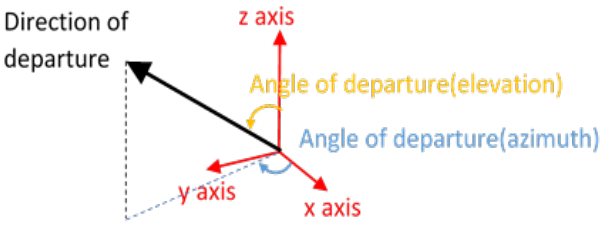

Figure 5: The angles of departure in azimuth and elevation planes.

There are some secondary characteristics which can be determined from geometrical properties of a ray and are described below.

1. **Delay**: The delay (τ) is defined as the ratio of the path length (d) and the speed of light (c).

$$\tau = \frac{d}{c}. \tag{1}$$

2. **Path loss**: The path loss is affected by the path length. This effect can be

computed using Frii's transmission loss equation as given below

$$\text{Path Loss} = 20\log_{10}\left(\frac{4\pi d}{\lambda_0}\right), \tag{2}$$

where λ_0 is the wavelength of the carrier frequency of the wireless communications system and d is the path length.

3. **Phase shift**: The phase shift of a ray is taken to be 180 degrees for every reflection. For a second order reflection, this phase shift would be 360 degrees. Phase is also affected by the Doppler effect which occurs due to mobility of communicating nodes.

4. **Doppler effect**: The Doppler effect is a change in frequency of a wave perceived by the observer when there is a relative motion between the observer and the source [5, Pages 352 - 358]. The phase of the received signal is affected by relative velocity between two nodes. Doppler effect is observed when the nodes are moving. The change in frequency, Δf is given by

$$\Delta f = \frac{\Delta v}{c} f_0, \tag{3}$$

where c is the velocity of light, Δv is the relative velocity between Rx and Tx, and f_0 is the frequency of operation.

Using the change in frequency from (3), the phase shift due to the Doppler effect is given as

$$\Delta\phi = \Delta f \times \tau, \tag{4}$$

where $\Delta\phi$ is the phase shift and τ is the delay.

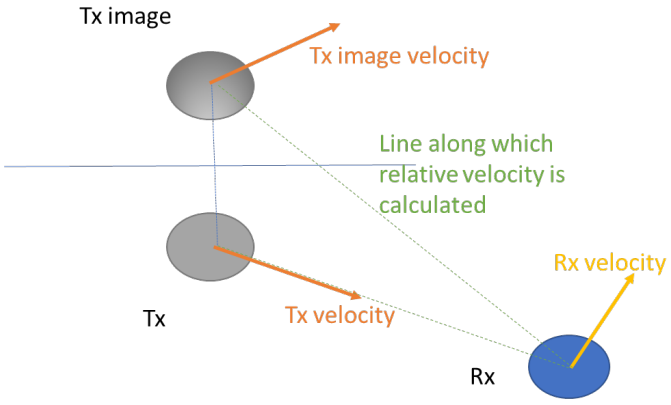

Figure 6: Schematic diagram to calculate the relative velocity of the reflected images for the scenario where two nodes are moving.

The Doppler effect will vary for different reflections because it is dependent on relative velocities of the reflected images but not on the actual node velocities. The relative velocities of the reflected images are calculated by

- first computing the velocities of the individual nodes
- and then finding the velocities of the reflected images recursively in the chronological order. The velocity of the node and the velocity of its image are opposite to each other about the normal vector of the reflection plane and constant along the reflection plane as shown in Fig. 6. Once the final image is obtained, then the relative velocity is calculated along the line joining the final image and the other node.

2.3 Block Diagram of NIST Q-D channel realization software

The NIST Q-D channel realization software has been developed to combine the deterministic results computed by raytracing with that of stochastic models of measurements conducted by the NIST [4]. The flow of this procedure is shown in Fig.7 where

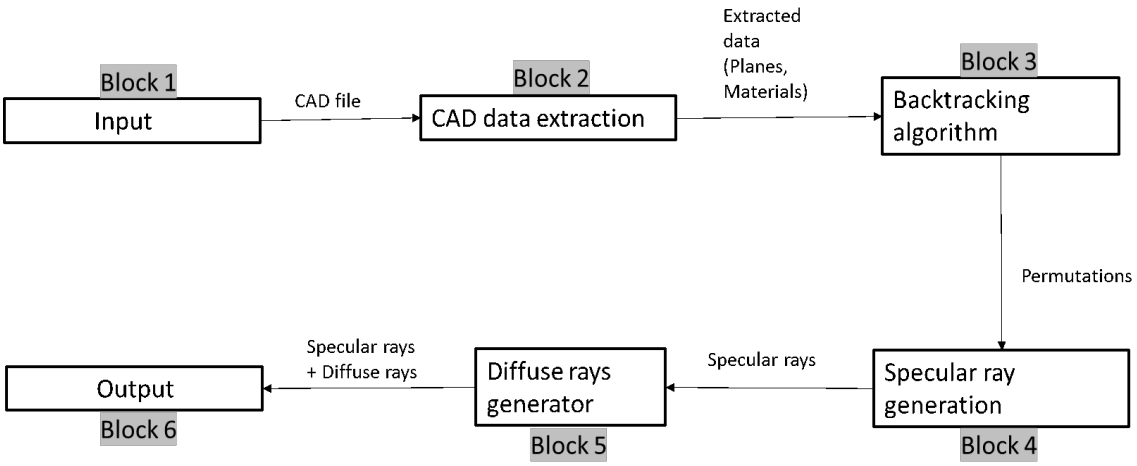

Figure 7: Step-by-step procedure of the NIST Q-D channel realization software.

Block 1 : Input Input parameters such as node position, velocities, order of reflection etc are fed into the software. The detailed information regarding the inputs can be found in Section 3.

Block 2 : CAD data extraction The way to process a CAD file is to extract the convex planar polygon data and compute its plane equations. Mobility model is included in this block. The detailed information about extracting the CAD data file can be found in Section 4.

Block 3 : Backtracking algorithm - All the possible permutations of plane combinations, where there is a possibility of reflection, are generated using backtracking algorithm. The detailed algorithm is described in Section 5.

Block 4: Method of Images - Once the plane combinations are known, the Method of Images [6] is used to compute the individual rays, as described in Section 6.

Block 5 : Q-D model - The stochastic channel model can be applied on every ray generated by the method of images to create a cluster of diffuse rays surrounding the specular ray. This cluster of rays is then generated as an output by the software. The detailed information can be found in Section 7.

Block 6 : Output - The final output consists of ray's attributes namely path loss, delay, phase, AoA, AoD at every time step and node combination. Output consists of node postions at every time step. Please refer to Section 8 for detailed information.

The following subsections describe the file and code structures of the NIST Q-D channel realization software which will be used in Sections 3-8.

2.4 File structure

The file structure is shown in Fig. 8 where *QDSoftware* is the main folder. The main folder contains two sub-folders as well as a main file *main.m*, two parameter configuration function files, i.e., *parameterCfg.m* and *nodeProfileCfg.m*. Two preexisting sub-folders are *Input* and *Raytracer*. The scenario sub-folders will be created when new scenario is generated either automatically by software or manually by user. It is worth noting that the working root folder must be QDSoftware\.

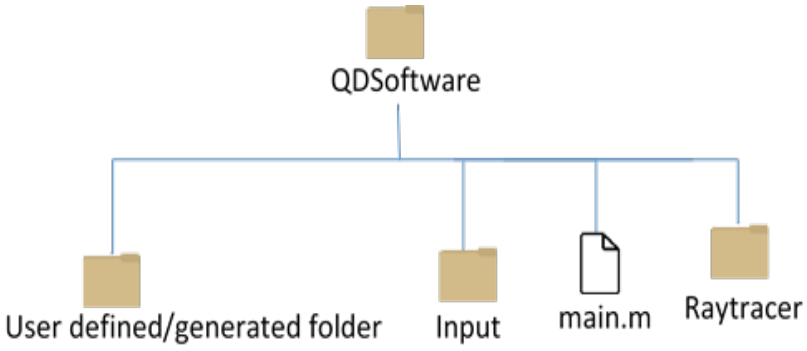

Figure 8: File structure of the NIST Q-D channel realization software

2.5 Code structure

There are four major MATLAB functions which call other functions to import the input and generate the output, namely the *main.m*, *parameterCfg.m* and *nodeProfileCfg.m*

in main-folder *QDSoftware*, and the *Raytracer.m* in sub-folder *Raytracer*. The *main.m* calls the *parameterCfg* function to import the system input parameters such as the number of nodes, the number of time steps, and the order of reflection from from files present in the *Input* folder and return them to the main file. The *main.m* also calls the *nodeProfileCfg* function to import the node profile information such as the node positions and velocities from files present in the *Input* folder and return them to the main file. The main file sends all the above mentioned input parameters to *Raytracer* function for further processing.

The *Raytracer* function first extracts the CAD information as shown in Fig. 9. Next, the first loop is to iterate through all the time steps. It is important to note that the node positions are updated for every iteration.

After that, a second nested loop is called, which iterates through all the node combinations. For every iteration, LOS ray is computed and its existence is verified. A third nested loop iterates through the order of reflection. Here, the backtracking algorithm and method of images are used to compute the rays. Once all the iterations are completed, the final output is stored in the folder *Output*.

The detailed description of each block used in NIST Q-D channel realization software is given in the following Sections.

3 Input (Block 1)

The first block of the NIST Q-D channel realization software is *Input*, which operates the system parameter configuration in *parameterCfg* function and the node profile configuration in *nodeProfileCfg* function.

The *parameterCfg* function is called to import several important system parameters as listed below from customized input configuration file *paraCfgCurrent.txt* in

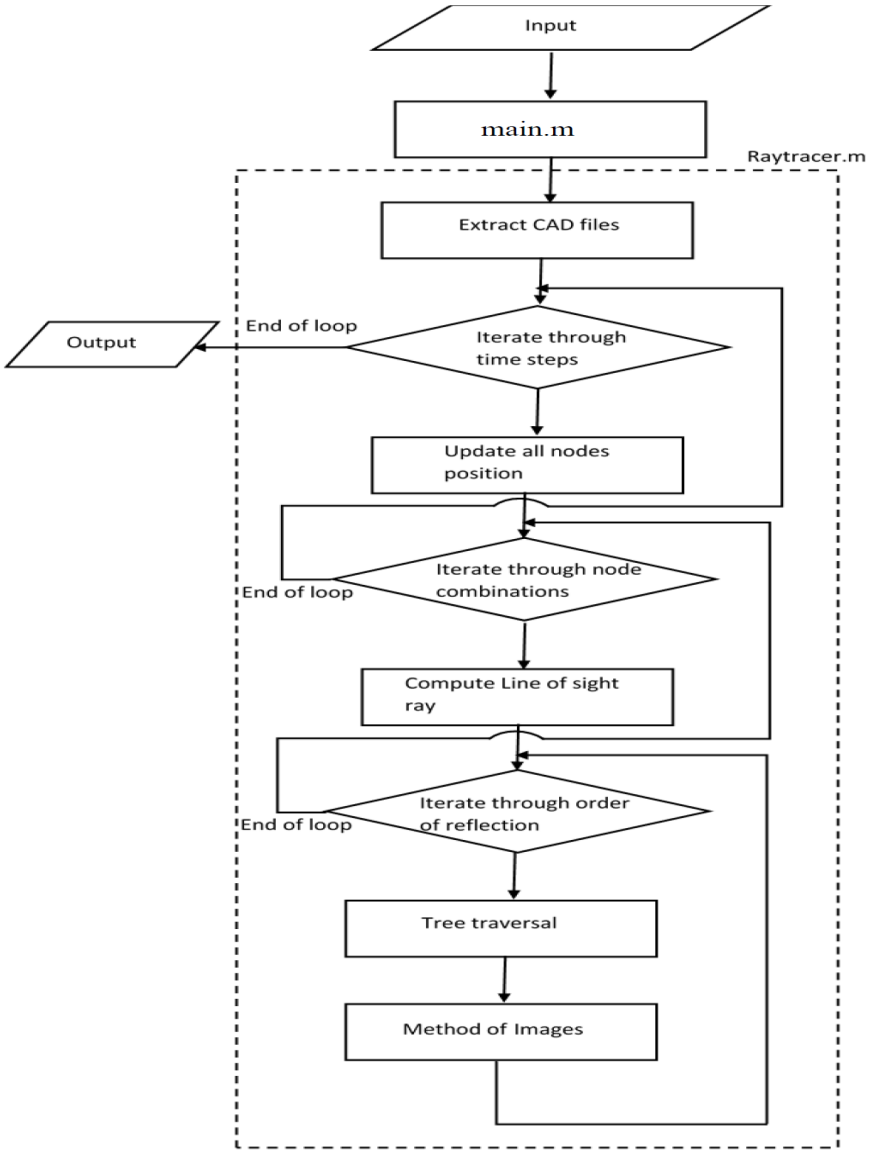

Figure 9: Flowchart of raytracer.m

folder *Input* and includes them in structure *paraCfg* to return to the main file. If the *paraCfgCurrent.txt* does not exist, a default input configuration file *paraCfgDefault.txt* in folder *Input* will be used.

Specifically, the system input parameters imported through file *paraCfgCurrent.txt* are defined as members in structure *paraCfg* as described below.

1. ***environmentFileName*** - This parameter defines the file name of the environment to be setup in the system.

2. ***generalizedScenario*** - This parameter defines the generalized scenario flag in the system. A generalized scenario is a scenario which cannot be classified as purely indorr or outdoor scenario. For instance, building where there are nodes both inside the building and outside the building. The default setting is $generalizedScenario = 1$.

3. ***indoorSwitch*** - This parameter defines indoor switch flag in the system. The default setting is $indoorSwitch = 1$.

4. ***inputScenarioName*** - This parameter defines customized input scenario name in the system. The user is required to enter this as string value when prompted via MATLAB command line.

5. ***mobilitySwitch*** - This parameter indicates whether mobility is present or not in the given scenario. If $mobilitySwitch = 1$ specifies that the nodes are mobile. On the other hand, $mobilitySwitch = 0$ specifies that the nodes are static or stationary.

6. ***mobilityType*** - This parameter defines the mobility model where $mobilityType = 1$ indicates linear mobility and $mobilityType = 2$ indicates custom mobility.

7. ***numberOfNodes*** - This parameter defines the number of nodes present in the network.

8. ***numberOfTimeDivisions*** - This parameter defines the total number of time steps. If $numberOfTimeDivisions = 100$ and $totalTimeDuration = 10$, then we have 100 time divisions for 10 seconds. Each time division is 0.1 secs in length.

9. ***referrencePoint*** - This parameter defines the referrence point of the center of limiting sphere. The default setting is *referrencePoint* = [3,3,2].

10. ***selectPlanesByDist*** - This parameter defines the selection of planes/nodes by distance. *selectPlanesByDist* = 0 means that there is no limitation (Default).

11. ***switchQDGenerator*** - This parameter indicates whether the diffuse[3] components are generated or not while generating the specular[4] components. If *switchQDGenerator* = 1 then diffuse components are generated with specular components otherwise only specular components are generated.

12. ***switchRandomization*** - This parameter indicates whether the location and velocity of each of the nodes are randomly generated or not. If *switchRandomization* = 1 then the locations and the velocities are generated randomly. On the other hand, if *switchRandomization* = 0 the locations and the velocities are imported from the files present in the *Input* folder.

13. ***switchVisuals*** - This parameter defines the switch visual flag for the system. The default setting is *switchVisuals* = 0.

14. ***totalNumberOfReflections*** - This parameter denotes the highest order of reflection considered in the model. For example, *totalNumberOfReflections* = 2 means the model considers the LOS, the first and the second order reflections. The default setting is *totalNumberOfReflections* = 2.

15. ***totalTimeDuration*** - This parameter defines the time period in seconds for which the simulation must run when the nodes are assumed to be mobile. The default setting is *totalTimeDuration* = 1.

[3]The diffuse components are the secondary multipath components generated based on rough surface scattering model.

[4]The specular components are the primary multipath components which follow the law of reflection.

Apart from the input system parameters mentioned above, there are three input files that are present in the *Input* folder, as shown in Fig. 10. These node profile configuration are imported via *nodeProfile* function called in *main* file, including in structure *nodeCfg* to return to the *main.m*, along with an updated *paraCfg* at meantime. It is important to note that these files are not needed when the nodes are generated randomly when *switchRandomization* = 1 in *parameterCfg* function and *main* file.

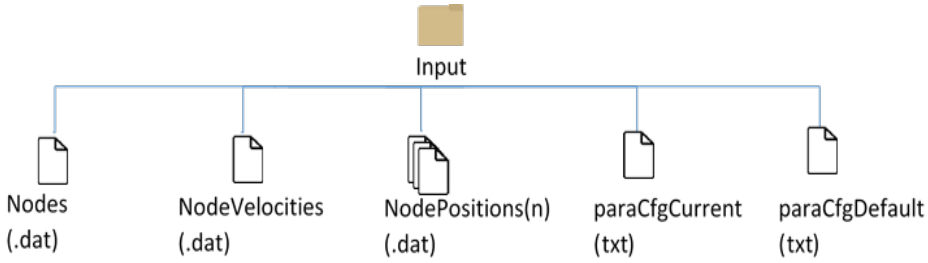

Figure 10: Files in *Input* folder.

The five input files that are present in the *Input* folder are described below.

1. ***nodes.csv*** - Each row in this file contains the (x, y, z) coordinate of the node. If the number of nodes is N, this file must have N rows.

2. ***nodeVelocities.csv*** - Each row in this file contains the node velocities in x, y and z directions. For N nodes, this file must have N rows. This file is needed only for linear mobility model when *mobilityType* = 1.

3. **NodePosition(n).csv** - In addition to *nodes.csv* and *nodeVelocities.csv*, the *Input* folder also contains the set of files named as *NodePosition1.csv*, *NodePosition2.csv*, upto *NodePositionN.csv* where N is the number of nodes. Each of the file contains the *numberOfTimeDivisions + 2* rows in the format of (x, y, z) indicating the positions of the particular node for the duration *totalTimeDuration*,

where ith row indicates the position at $(i-1)$ time step. This file is needed only for custom mobility model when $mobilityType = 2$.

Regarding the *Input* block, It is important to note that:

1. Create or enter the existed scenario name to process at command line. The user can enter any scenario name without space to indicate the type;

2. Always configurate all parameters in paraCfgCurrent.txt of the customized scenario folder, i.e.,
   QDSoftware\CUSTOMIZEDSCENARIO\Input\paraCfgCurrent.txt;
   if this *paraCfgCurrent.txt* file is not existed in this folder, the *paraCfgDefault.txt* will be load as default setting;

3. The *QDSoftware\Input* folder includes the source file of input data, it should be kept in root folder as always. Unless the user create the new scenario folder manually, every new scenario created by script will copy the input data automatically to it's folder from *QDSoftware \Input*, including the *paraCfgCurrent.txt* and *paraCfgBackup.txt* as a reference.

4 CAD Data Extraction (Block 2)

4.1 CAD Format

CAD file consists of geometrical information of a given scenario. These geometrical properties can be represented as point clouds, lines, and/or polygons. In NIST Q-D channel realization software, AMF file format is used as the standard-form of input CAD file format. AMF is a file format that is like STL format, i.e., geometrical structures are represented in terms of triangles only. In addition to geometrical properties,

AMF file format can also store material properties of each triangle. AMF file format is an XML based format and for a detailed description, please refer to [7]. The output of the CAD data extraction block would be the set of triangles which are described by the vertices in x, y, z format and the material properties.

For example, a cube has 6 sides and it is represented as a set of 12 triangles. Once

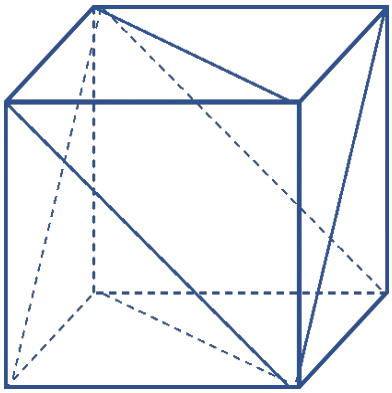

Figure 11: Cube in AMF file format described by 12 triangles.

we extract the three vertices of a triangle, we can construct its plane equation. The normal of this plane equation are given by right hand rule according to AMF format guidelines. If the normal is to be taken in clockwise direction then *IndoorSwitch* in *paraCfg* file should have a value of '1'. For counterclockwise direction this value should be '0'. These are shown in Fig.12 and Fig.13

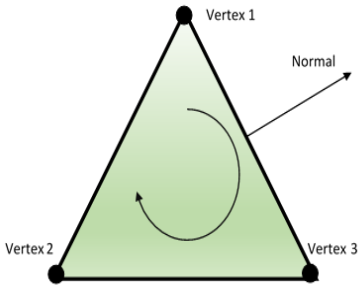

Figure 12: A triangle and its normal when *IndoorSwitch* has a value of '1'. The normal is taken in clockwise direction

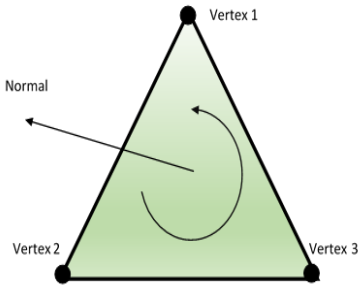

Figure 13: A triangle and its normal when *IndoorSwitch* has a value of '0'. The normal is taken in counterclockwise direction

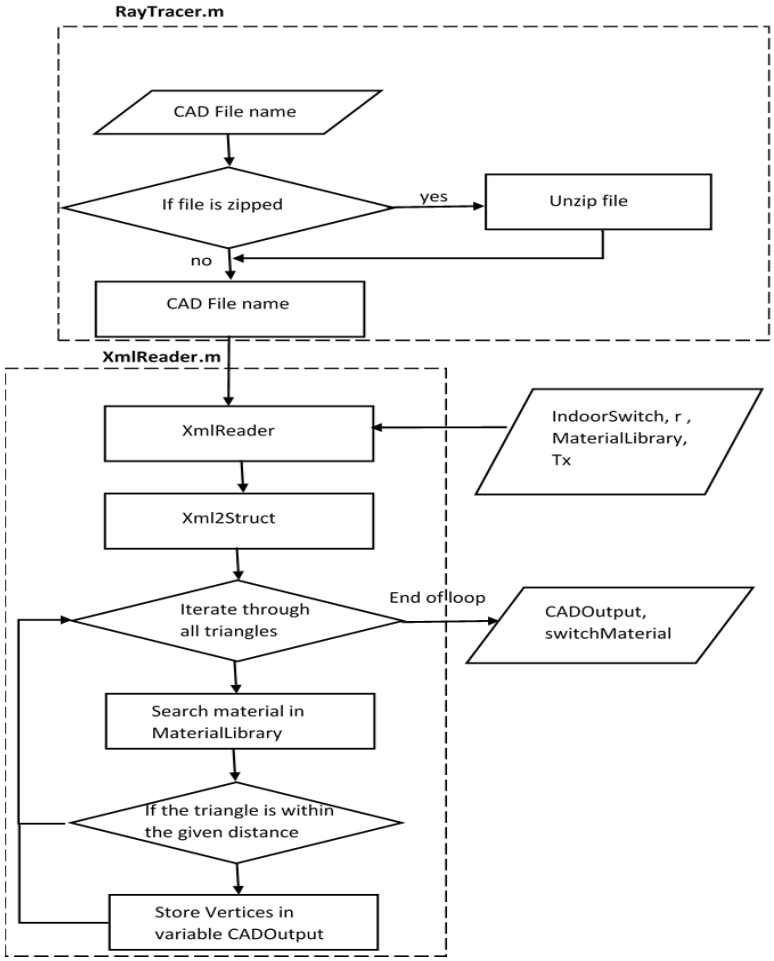

Figure 14: Flow chart of CAD extraction

4.2 Code Structure

Three functions *RayTracer.m, XmlReader.m* and *Xml2Struct.m* are used to extract the CAD information. *Raytracer.m* is the function which calls *XmlReader.m* function as

shown in flow chart in Fig. 14. *XmlReader.m* takes *IndoorSwitch*, *MaterialLibrary*, r and reference point as input parameters where *IndoorSwitch* defines how the normal is oriented. Normal of a triangle either comes out of the plane (vertices defined in clockwise direction using right hand rule) or into the plane (vertices are in anti-clockwise direction using right hand rule). *MaterialLibrary* is a table which has the material properties. The quantity r and the reference point denote the radius of limiting sphere and the center of the limiting sphere, respectively. These parameters are also described in section 4.3. in detail. The MATLAB *Xml2Struct.m* function extracts the Xml data and converts it into a struct variable s. The struct variable s is arranged in such a way that the triangles and the corresponding vertices are stored in a chronological order. We iterate through all the triangles and search if the material is present in the material library or not. If it is present, then it is associated with the triangle. The information of the triangles is finally stored in *CADOutput* variable. The *CADOutput* variable is a 2D array with 14 columns and the number of rows that is equal to the number of triangles. One row of *CADOutput* variable is shown in Fig. 15.

x1	y1	z1	x2	y2	z2	x3	y3	z3	a	b	c	d	Material Id

Figure 15: A row of *CADOutput* variable

For the first 9 columns, x1, y1, z1 are x, y, z coordinates of 1st vertex of the triangle; x2, y2, z2 are x, y, z coordinates of 2nd vertex of the triangle, and x3, y3, z3 are x, y, z coordinates of 3rd vertex of the triangle. A plane equation is given as $ax+by+cz+d = 0$. The variables a, b, c, d in the columns 10 to 13 are x, y, z coefficients and the constant, respectively. The variable 'Material Id' in last column helps in identifying the material associated with the triangle.

4.3 Omitting Triangles based on Distance from a Reference Point

The function *distanceLimitation.m* is used to omit the triangles which are far from the defined reference point. This is because the rays associated with the triangles would have very large path loss, which does not contribute significantly in the total received power. If we consider these triangles, it would also increase the computational time significantly. The problem statement is now to find the triangles which contribute significantly in the total received power. This can be done by intersecting the triangles and the sphere[5] of radius r considering the reference point as the center of the sphere. The triangles that intersect the sphere are considered while eliminating others. There are three possible cases in which a triangle intersects the sphere and are described below.

Case 1 : Either of the three vertices are within the sphere

To know whether the triangle intersects with the sphere, we measure the distance between each vertex and the center of the sphere. If at least one distance is less than or equal to the radius of the sphere, then the triangle lies within the sphere as shown in Fig. 16.

Case 2 : A triangle whose vertices lie outside the sphere and the projection of the center of the sphere onto the plane containing the triangle lies within triangle

When the distance between the center and projected point is less than or equal to radius, then the triangle lies within the sphere as shown in Fig. 17.

Case 3 : A triangle whose vertices lie outside the sphere and the projection

[5]Sphere is the locus of all points whose distance (radius) from a given point (center is the reference point) is constant.

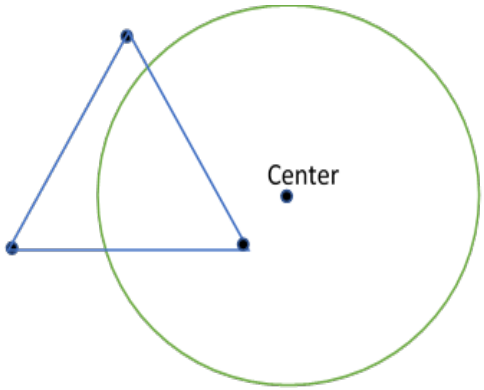

Figure 16: Case 1: Either of the three vertices are within the sphere.

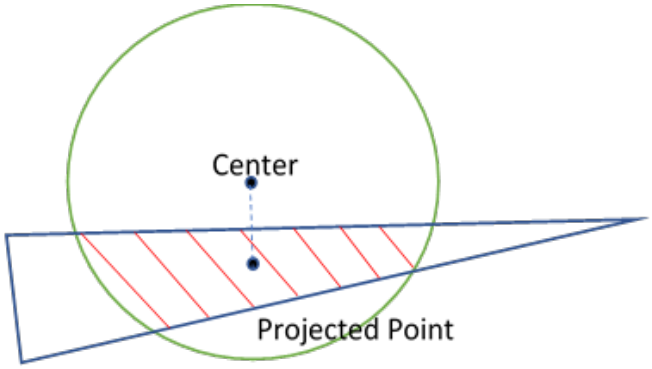

Figure 17: Case 2 : A triangle whose vertices lie outside the sphere and the projection of the center of the sphere onto the plane containing the triangle lies within triangle.

of the center of the sphere onto the plane containing the triangle lies outside triangle

The perpendicular distance[6] of center of the sphere to each of the three sides is calculated. If this perpendicular distance is less than or equal to the radius of the sphere, then the triangle lies within the sphere as shown in Fig. 18.

[6]Using the Pythagoras theorem, the perpendicular distance of center of the sphere from each of the sides can be calculated as the square root of the sum of square of projected distance and perpendicular distance of projected point to the side.

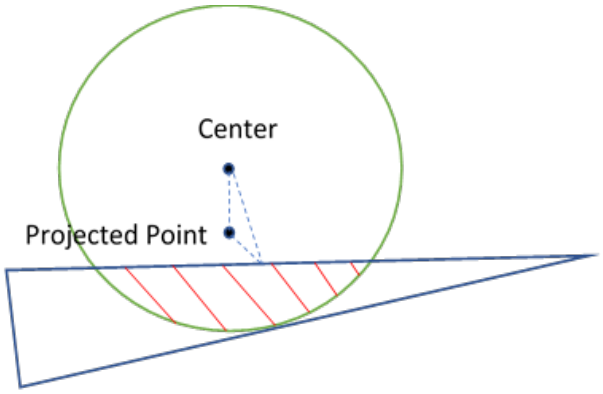

Figure 18: Case 3 : A triangle whose vertices lie outside the sphere and the projection of the center of the sphere onto the plane containing the triangle lies outside triangle.

5 Backtracking Algorithm (Block 3)

Backtracking algorithm is a special case of depth first search algorithm that is used to find all the possible permutations by constructing a recursive tree. The set of all entities form the first generation of this tree, e.g., $\{1, 2, 3\}$ as shown in Fig. 19. For every element in the first generation, the entire set becomes the second generation. This recursively takes place until the nth generation of the tree. Every path that traverses from the first generation to the last generation gives one combination as shown at the bottom of Fig.19. Backtracking algorithm is used to find all the possible solutions of a specific problem, such that the candidates are incrementally built using depth first search algorithm while candidates without a solution are abandoned [8, Pages 272-275]. At every traversal through a node, the backtracking algorithm checks for a given condition to be satisfied and if it fails then the subsequent traversal through this node is avoided.

Rather than performing raytracing on every possible combination of reflecting surfaces, it would be computationally efficient to remove the combinations where reflection is impossible. This can be achieved through the backtracking algorithm [8].

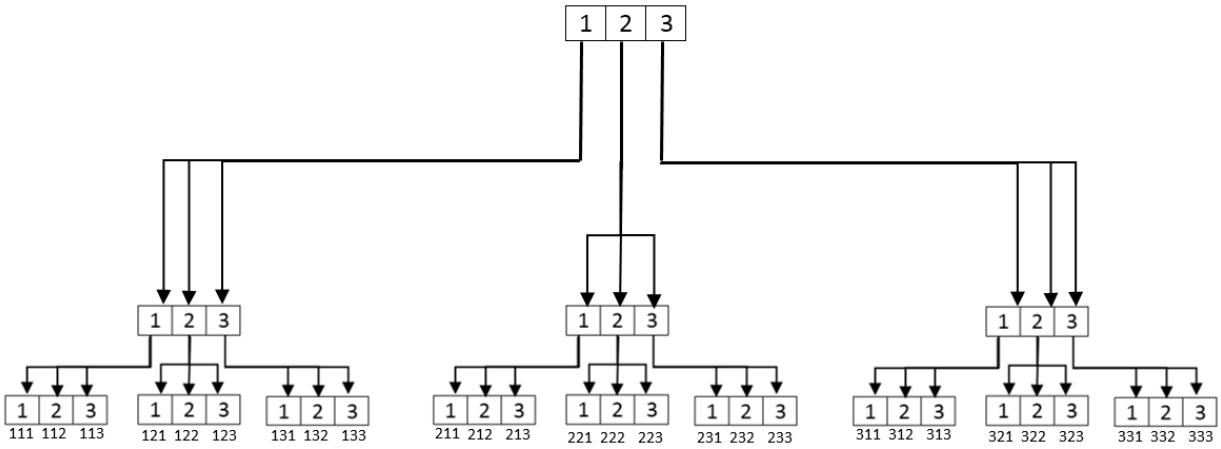

Figure 19: A recursive tree consists of three generations for a set 1,2,3. The last line shows all the combinations after traversing through every path in the tree.

Once the information from CAD file is extracted, i.e., the triangles and plane equations, the information must be processed such that the reflections can be computed. For a reflection to occur, a ray must be incident on the face of the triangle. For an nth order reflection, the ray reflects n times from triangles (not necessarily distinct). All the triangle combinations are computed to find reflected rays. Tx is the first node of the tree and Rx is the last node of the tree. The entire set of triangles extracted from the CAD file is the second generation of the tree. Further, the child nodes of each element in the second generation are again represented by the complete set of triangles. This process of generating child nodes carries on n number of times that is equal to the order of reflection. In the last generation, all the nodes end at the Rx.

As shown in Fig. 7, the backtracking algorithm is the third block where backtracing is performed on CADOutput variable generating by the block 2. Note that the CADOutput variable consists the information extracted from the CAD file. An illustrative example to explain this backtracing method is described below:

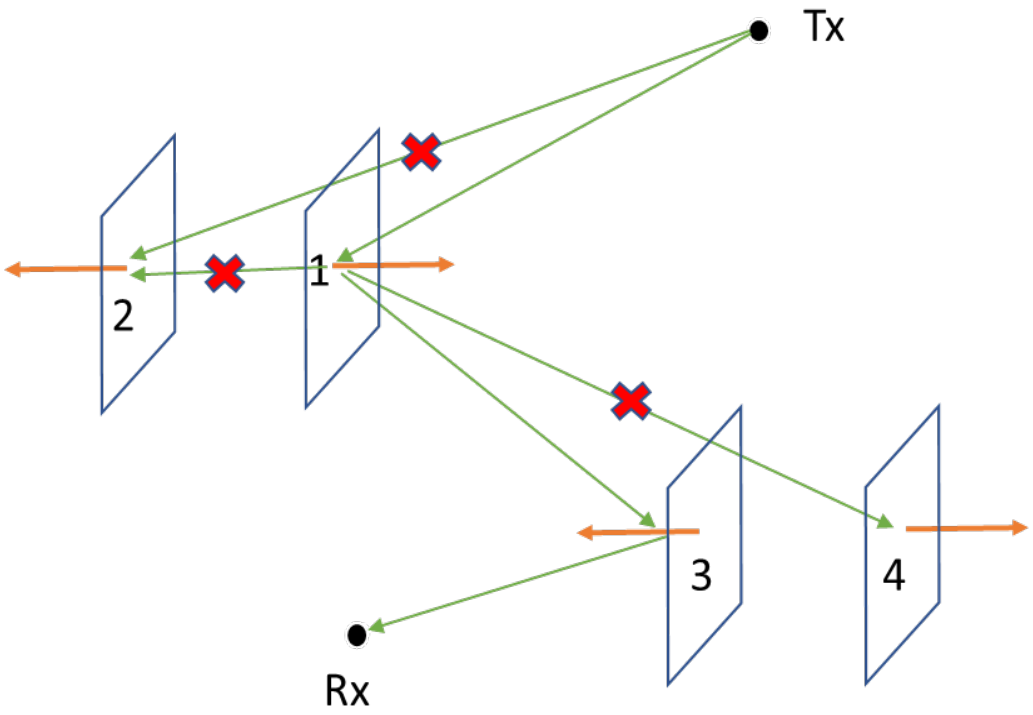

Figure 20: A scenario with 2 planes and a pair of transmitter (Tx) and receiver (Rx).

Step 1: Consider Tx as the first node or parent node of the tree as seen in Fig .20.

Step 2: Consider Rx as the last node of the tree as seen in Fig. 20.

Step 3: The first generation of child nodes is the complete set of planes (CADOutput). The set of child nodes at each node of first generation is the complete set of planes as seen in Fig. 21.

Step 4: A link does not exist between the identical nodes present in the first generation and the second generation, because a ray cannot reflect from the same plane twice. Such cases are eliminated as shown in Fig. 21.

Step 5: A link does not exist between two planes for the cases when (a) they face away from each other (b) they have same plane equation and (c) the dot product between the normal of the planes is positive. Such cases are also eliminated. In the above

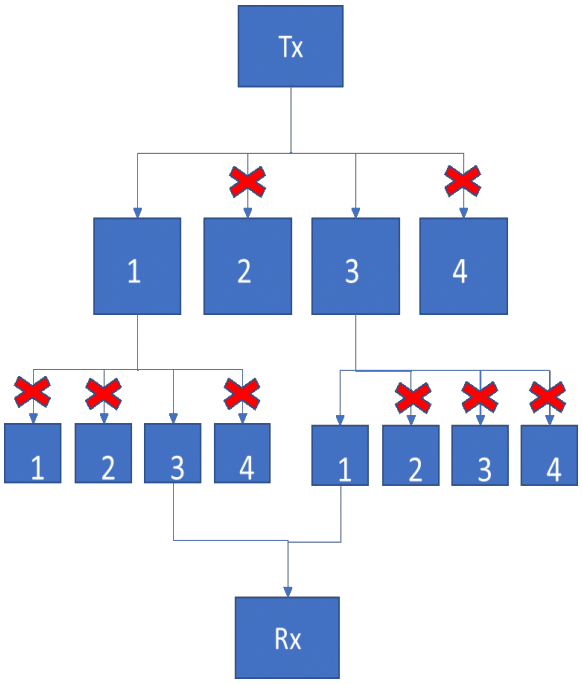

Figure 21: Schematic diagram of a Tree describing the backtracking algorithm.

example it can be clearly seen that a ray does not exist between planes 1 & 2, 1 & 4, 3 & 4, 3 & 2, Tx & 2, Tx & 4, 2 & Rx, and 4 & Rx.

Step 6: The final output is the set of all the combinations of the planes where reflection is possible.

Complexity: the worst-case time complexity is $\mathcal{O}(n^m)$ where n denotes the total number of planes and m represents the highest order of reflection.

5.1 Code Structure

Backtracking algorithm is implemented using *Treetraversal.m* and *VerifyTreeTraversalPath.m* functions. *CADOutput*, i.e., CAD information, generated from previous step is given as input to the function *Treetraversal.m*. A loop is run to iterate through all *CADOutput* rows. This means that we are iterating through all the triangles present in the CAD file. At every iteration, the possibility of ray is probed. If a ray does not exist,

then we move onto the next iteration. If there is any possibility of ray to exist, then *ArrayOfPlanes, ArrayOfPoints* and *ArrayOfMaterials* variables are populated. The variable *ArrayOfPlanes* contains the plane equations of the triangles, *ArrayOfPoints* contains the vertices of the triangles, and *ArrayOfMaterials* contains the material properties of the triangle. Once all the variables are populated, we recursively perform the same task for N times, where N denotes the order of reflection.

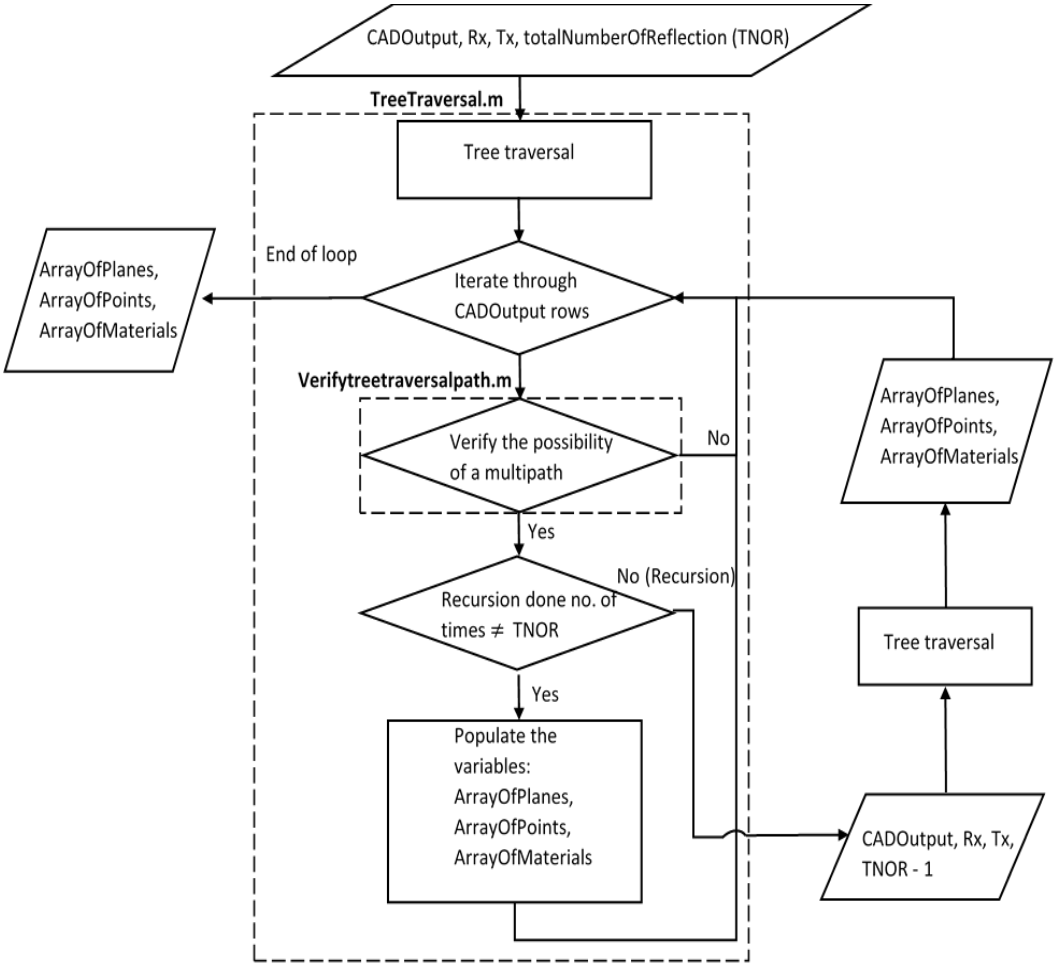

Figure 22: Flow chart of backtracking algorithm.

The row format of variables *ArrayOfPlanes, ArrayOfPoints* and *ArrayOfMaterials* are shown in the Figs. 23, 24 and 25, respectively.

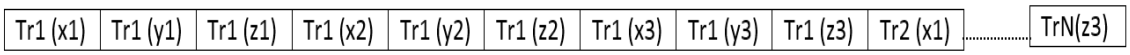

Figure 23: A row of *ArrayOfPoints* variable.

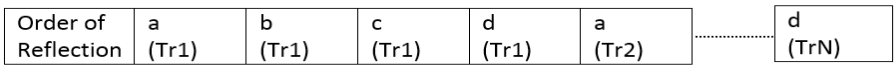

Figure 24: A row of *ArrayOfPlanes* variable.

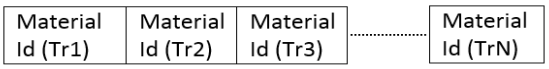

Figure 25: A row of *ArrayOfMaterials* variable.

In Figs.23-25, $\{Tr1, Tr2, \cdots, TrN\}$ represent a set of triangle combinations, where N is equal to the order of reflection. The parameters $(x1, y1, z1)$ are the x, y and z coordinates of 1st vertex of the triangle; $(x2, y2, z2)$ are x, y and z coordinates of 2nd vertex of the triangle, and $(x3, y3, z3)$ are x, y and z coordinates of 3rd vertex of the triangle. From the vertices of the triangle, a plane equation is constructed and is given as $ax + by + cz + d = 0$, where a, b, c are x, y, z coefficients, respectively, and d is a constant.*MaterialId* variable in *ArrayOfMaterials* helps in identifying the material associated with the triangle.

6 Method of Images (Block 4)

The method of images is based on geometrical optics. According to the principles of geometrical optics, in free space a light ray travels in a straight line which has

minimum path-length between two points. This phenomenon is known as *Fermat's principle* [9, Pages 134 - 141]. In case of reflection, the path-length of the ray should be minimum between two points while the ray is incident on a plane surface. This problem is known as *Heron's Problem* [10], which can be solved by taking the image $P1'$ of a point $P1$ on to the plane surface R and connecting the other point $P2$ to this image via a line as shown in Fig. 26. The intersection point M of this line with the plane is again connected to the point $P1$.

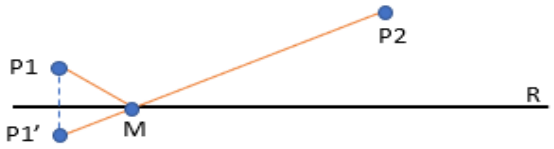

Figure 26: Reflected path between points $P1$, $P2$ and reflecting plane R.

The method of images is a raytracing technique that provides an exact solution as opposed to an approximate solution based on trial and error methods, such as shoot and bounce method. The method of images, however, can only be used for reflections whereas the other methods e.g., shoot and bounce can be used to compute reflections along with diffraction and refraction [6]. Once the set of combinations of planes is obtained, then the raytraced path for every combination is computed using the method of images.

An example is presented below to explain the method of images to compute the second order reflection as shown in Fig. 27.

Step 1: Import Tx and Rx locations along with a set of planes as input parameters.

Step 2: Compute image of Tx in Plane 1 which is the first recursion.

Step 3: Compute the image of Tx image in plane 2 which is second recursion.

Step 4: Next, the vector joining the image of Tx image with Rx is calculated where the length of this vector is the path-length.

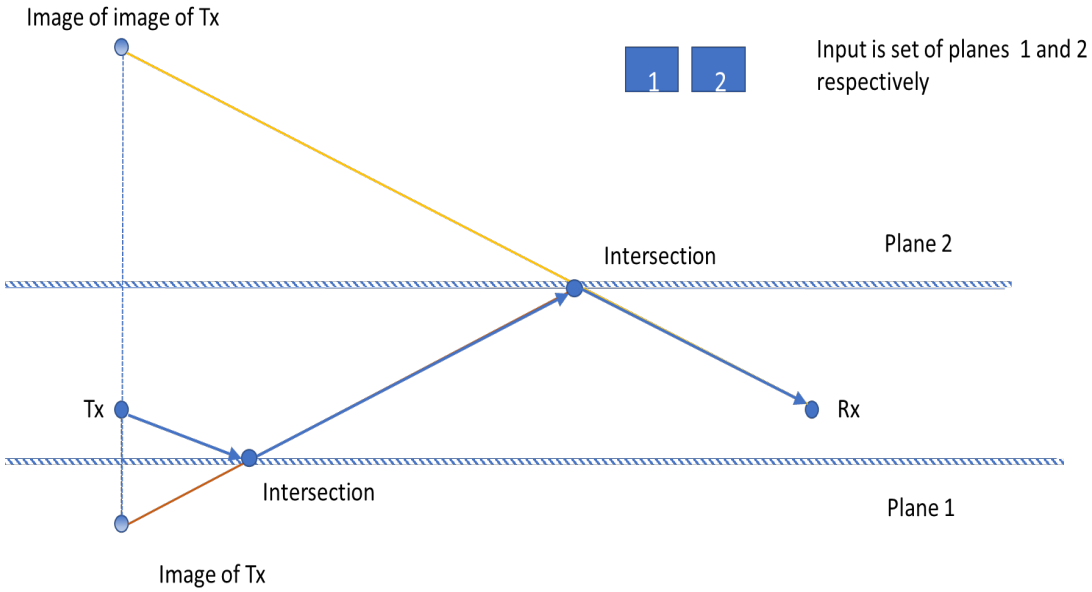

Figure 27: Method of images for a scenario with an input of two planes.

Step 5: The intersection of this vector with plane 2 is the point of incidence on plane 2 for the reflected ray. By connecting the intersection with Rx, the direction of arrival (DoA) vector is obtained as shown in Fig. 27.

Step 6: A vector is formed by connecting the intersection of the ray with plane 2 and image of Tx. The intersection of this vector with plane 1 is subsequently computed.

Step 7: The intersection on plane 1 is connected to intersection on plane 2, also forms a vector. This vector is a part of the reflected ray.

Step 8: The Tx is connected to the intersection on plane 1. This gives the direction of departure (DoD) vector.

Complexity: As mentioned earlier, $\mathcal{O}(n^m)$ is the total time complexity to traverse through all the combinations of planes as described in backtracking algorithm. In addition, there are total $(m+1)$ vectors generated through the method of images and each vector is verified for its intersection with any of the planes. This entire process takes $\mathcal{O}(n^{m+1})$ steps to complete. Therefore, the total complexity is $\mathcal{O}(n^{2m+1})$, which

can be obtained by multiplying the complexities of both algorithms.

6.1 Code Structure

The method of images algorithm is implemented using the functions: *multipath.m, singlepathgenerator.m* and *LOSOutputgenerator.m*. In addition to these functions, *verifyPath.m* function is also crucial for functioning of method of images.

Note that *multipath.m* generates all the possible rays for given locations of Tx and Rx (*Tx, Rx* in Fig. **??**) and order of reflection. It iterates through the output of tree traversal part of the code, i.e., *ArrayOfPlanes* and *ArrayOfPoints*. A single row of *ArrayOfPlanes* and *ArrayOfPoints* is taken and passed through the *singleMultipathGenerator.m* to generate a ray. Once we generate a ray, we compute delay, gain, AoA, AoD and phase. Further, the Q-D model is applied to the ray generated through *singleMultipathGenerator.m*.

The method of images is implemented in *singleMultipathGenerator.m* which takes input from *multipath.m*. Next step is to extract the plane equations and images of *Tx* along these extracted planes. As shown in Fig. 29, *singleMultipathGenerator.m* is called recursively for N times, where N is the order of reflection. The final image of *Tx* is then connected to the *Rx*. This forms the DoA vector. We also get path-length in this step. In the next step, we compute the point where this DoA intersects with the plane where this reflection has happened. In the last recursion, DoD is assigned the value of DoA and when we exit from subsequent recursions, DoD is dynamically updated with the newly computed ray vectors which are a part of the reflected ray. We also check if DoD intersects with any other triangles in the CAD file using *verifyPath.m*. Note that the path between Tx and Rx exists when there is no intersection with any of the triangles. If the path exists, then we exit from this recursion and we enter into

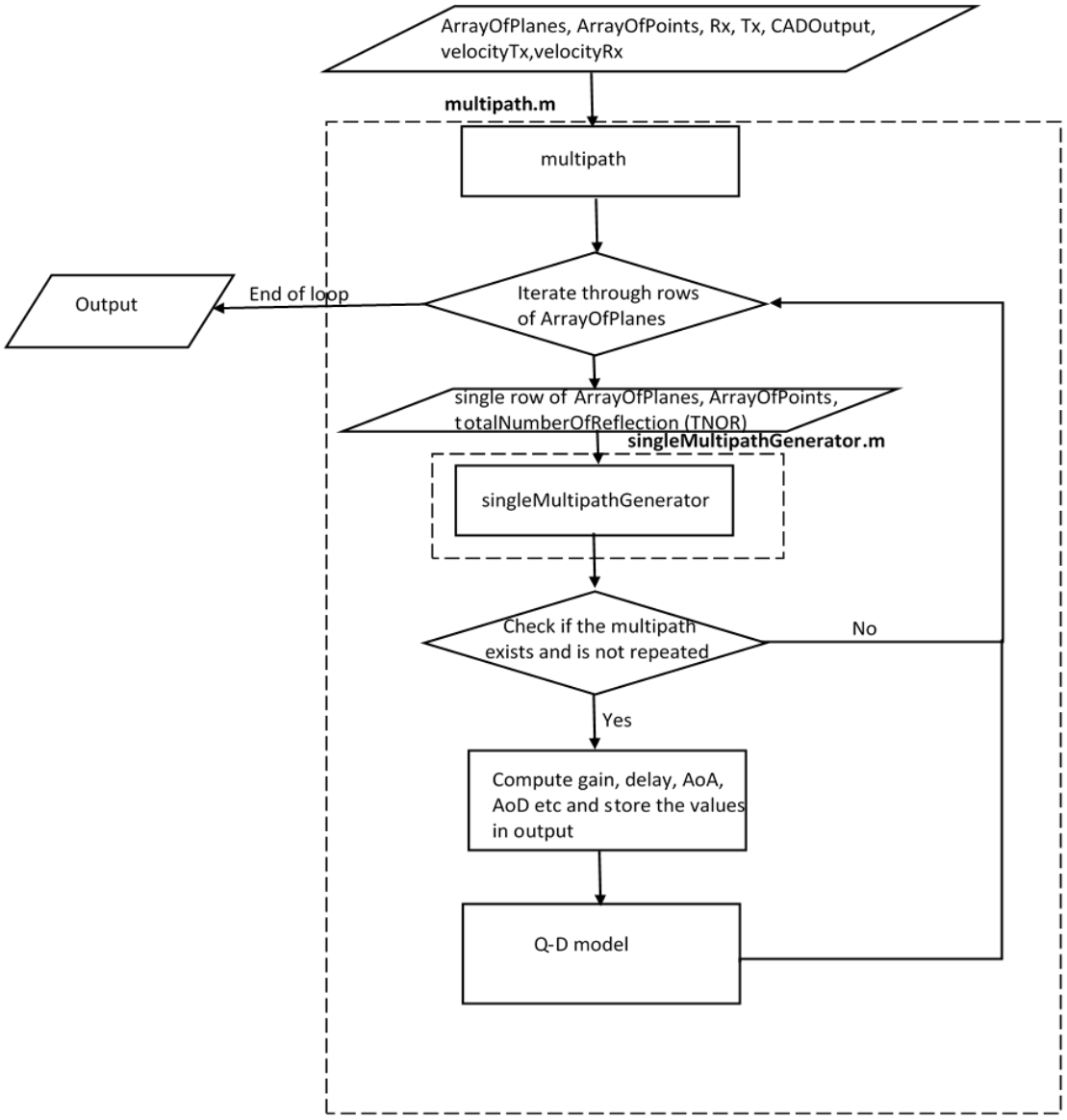

Figure 28: Flow chart of *multipath.m*.

the function which recursively called the previous function. DoD is again calculated by joining the intersection point which was calculated in the previous call with the reflected image obtained in the present function. We again check if this DoD intersects with any of the other triangles or not. This process continues until there is a intersection. In the end, if there is no intersection with any of the triangles, the complete ray between Tx and Rx is finally generated.

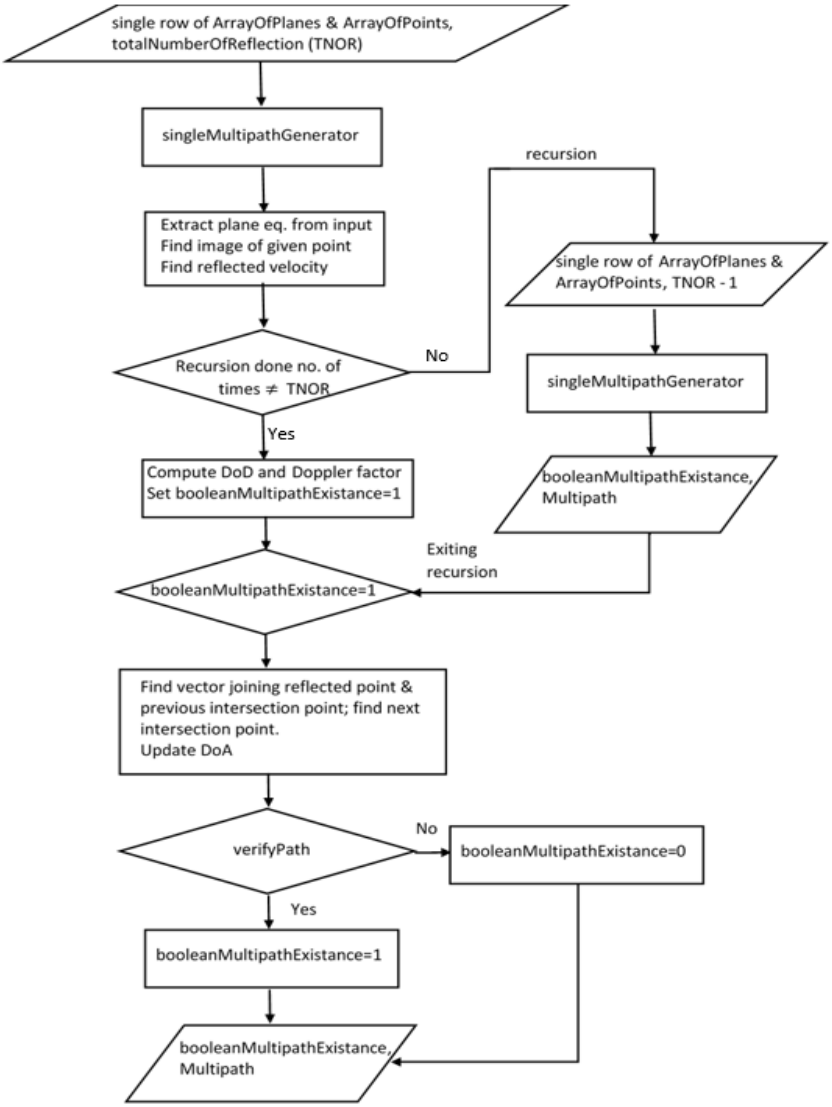

Figure 29: Flow chart of *singleMultipathGenerator.m*.

Based on the ray generated above, we compute various parameters such as gain, delay, AoA, AoD and phase. The delay is obtained by dividing the path-length by the speed of light. The gain is computed by using Frii's transmission formula (c.f. (2)). The AoA and AoD vectors are computed by obtaining the coordinates in spherical coordinate system. The theta and phi of DoA and DoD correspond to AoA and AoD respectively. For every reflection, a phase shift of 180 degrees is added. Note that the

phase of the ray changes by 180 degree when the ray reflects from the surface of a medium with higher refractive index than that of the medium in which it's travelling. The values obtained for these parameters are finally stored in *Output* variable.

Note that *LOSOutputgenerator.m* is a simple function which computes the line-of-sight (LOS) path for a given Tx and Rx locations. This is done by connecting *Tx* and *Rx*, and checking if it does not pass through any of the obstacles using *verifyPath.m*. It is similar to *singleMultipathGenerator.m* and is implemented in *Raytracer.m* function.

7 NIST Q-D Model (Block 5)

The NIST Q-D model is a rough surface scattering model that is developed using the channel measurements conducted in a lecture room. In this model, the rays computed by the method of images are considered as the specular (deterministic) rays. These specular rays are the primary multipath between the Tx and the Rx. There are other secondary rays, which are generated due to the scattering of the specular reflected rays from objects. These secondary rays are called as *diffuse* rays. Due to scattering, they form clusters around the reflected specular rays. These clusters are characterized by the parameters such as angular spread, delay spread, etc. Every cluster is composed of

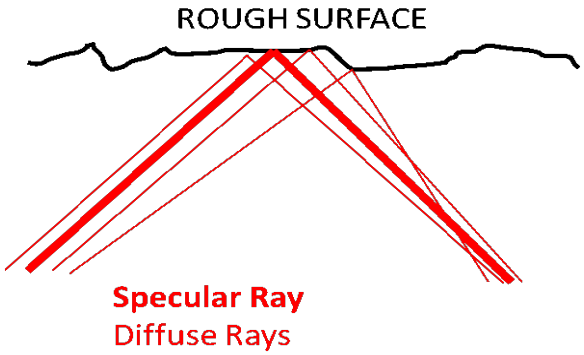

Figure 30: Rough surface scattering of specular ray (thicker red line).

a specular component and multiple diffuse components. K factor is defined as the ratio of specular power after scattering to the sum of the total cluster power. Note that the K factor gives a notion of how much power is scattered from the specular component. The power levels of diffuse components exponential decrease from the specular component. This rate of decay is given by γ. The delay spread can be modeled using the Poisson distribution with mean and variance λ. Angular spread follows a Laplacian distribution whose mean is equal to the angle of specular component and variance is given by Θ. The parameters K factor, γ, λ, Θ and reflection loss are randomly generated using the Normal distribution which is characterized by the mean and standard deviation obtained from channel measurements conducted at NIST.

7.1 Code Structure

In our Q-D realization software, a total of 20 diffuse multipath components (MPCs) are generated for each of the deterministic rays. The diffuse components are further divided into the precursor and postcursor components. The precursor components arrive before the deterministic component and the postcursor components arrive after the deterministic component. Out of 20 diffuse multipath components, there are 3 precursor components and 17 postcursor components. We are considering 3 precursor components since the total precursor components are far lesser in number. It is because of the fact that the probability of a precursor arriving earlier than the specular component decreases rapidly as the difference between their delays increases. The rationale behind considering 17 postcursor components is that the power of postcursor components becomes insignificant enough to be neglected after 17. It is worth noting that one can also consider different number of precursor and postcursor components.

Material librarys attributes are given below:

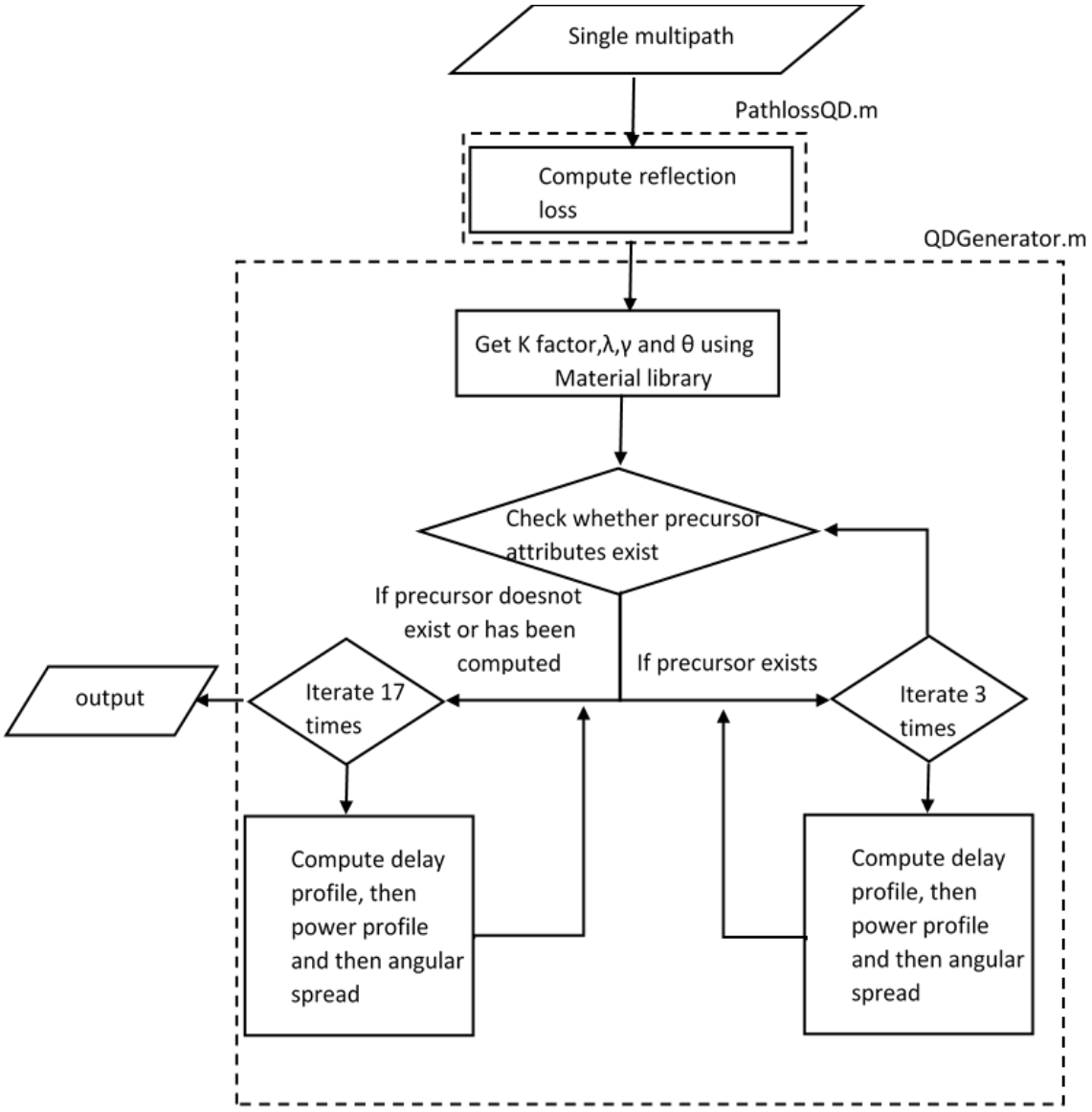

Figure 31: Flow chart of NIST Q-D model.

- **PrimaryKey** - This defines a row number of the material library table. Throughout the code, this serves as a *MaterialId*.

- **Reflector** This defines the name of the material.

- **sigma_k_Precursor** This is the standard deviation of K factor for the precursor components.

- **mu_k_Precursor** This is the mean of K factor for the precursor components.
- **sigma_k_Postcursor** This is the standard deviation of K factor for the postcursor components.
- **mu_Y_Postcursor** This is the mean of γ for the postcursor components.
- **sigma_Y_Precursor** This is the standard deviation of γ for the precursor components.
- **mu_Y_Precursor** This is the mean of γ for the precursor components.
- **sigma_Y_Postcursor** This is the standard deviation of γ for the postcursor components.
- **mu_lambda_Precursor** This is the mean of λ for the precursor components.
- **sigma_lambda_Precursor** This is the standard deviation of λ for the precursor components.
- **mu_lambda_Postcursor** This is the mean of λ for the postcursor components.
- **sigma_lambda_Postcursor** This is the standard deviation of λ for the postcursor components.
- **mu_sigmaTheta** This denotes the mean of Θ.
- **sigma_sigmaTheta** This denotes the standard deviation of Θ.
- **mu_RL** This denotes the mean of reflection loss.
- **sigma_RL** This denotes the standard deviation of reflection loss.
- **DielectricConstant** This defines dielectric constant of the material.

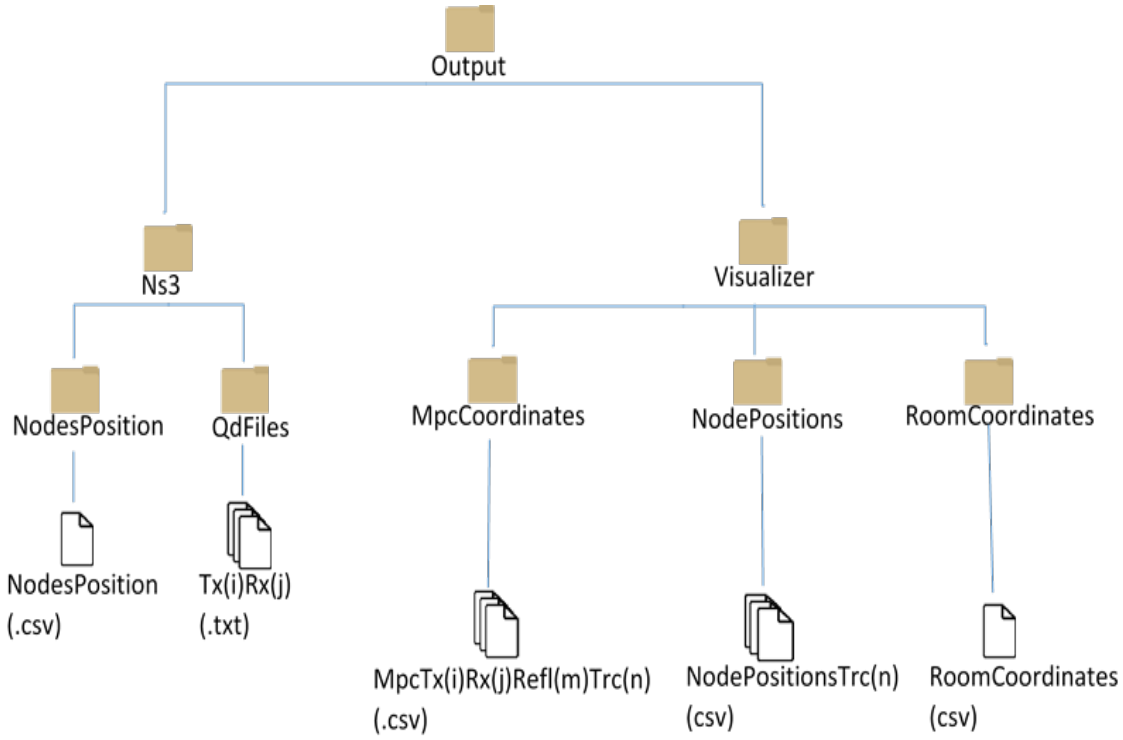

Figure 32: File Structure of *Output*.

8 Output (Block 6)

The final output consists of ray's attributes namely path loss, delay, phase, AoA, AoD at every time-step and Tx Rx node combination. Output consists of node positions at every time-step. Output also consists of *.csv files which aid in visualizing the rays. Output files are generated and saved in the *Output* folder which contains two sub-folders: *Ns3* and *Visualizer*. *Ns3* further contains two sub-folders: *NodesPosition* and *QdFiles*, which are described in detail below.

1. ***NodesPosition*** - this folder consists of *NodesPosition.txt* file. The file contains the node postions at first time interval. This is equivalent to *nodes.csv* file in *Input* folder.

2. ***Qdfiles*** - this folder consists of *Tx(i-1)Rx(j-1).txt* trace files for $(i-1)$th and $(j-1)$th node that can be directly used as input for NS-3. The format of this file for the scenario when the nodes are stationary is given as follows.

   (a) Number of rays occupies the first row

   (b) Delay of each ray is stored in the second row

   (c) Path gain of each ray is stored in third row

   (d) Phase offset of each ray is stored in fourth row

   (e) AoD (elevation) of each ray is stored in fifth row

   (f) AoD (azimuth) of each ray is stored in sixth row

   (g) AoA (elevation) of each ray is stored in seventh row

   (h) AoA (azimuth) of each ray is stored in eighth row

   Note that the above format is same for different time instances. However, this format is repeated for different time instances if the nodes are mobile. For example, the first time instance data is stored in first eight rows, the next eight rows stores second time instance data and so on.

Visualizer folder consists of three sub-folders - *MpcCoordinates, NodePositions, and Roomcoordinates* which are described below.

1. ***MpcCoordinates*** - This folder contains several *.csv files whose nomenclature is given by *MpcTx(i-1)Rx(j-1)Refl(m)Trc(n).csv* where i and j are node indices, m is the order of reflection and n defines the time instance. There are a total of $3 \times (m+2)$ columns in each file where the first three columns consist of x, y and z coordinates of the node, respectively. The last three columns in each file are the x, y and z coordinates of other communicating node. The remaining columns

have the coordinates of intersection points present on the reflecting planes. To model/ visualize the ray, one has to simply connect the coordinates present in first, second and third columns to that of the coordinates in fourth, fifth and sixth columns. The point coordinates present in the fourth, fifth and sixth columns is again connected to the next three columns. This process of connecting points must be repeated until we reach the last three columns.

2. ***NodePositions*** - This folder consists of *NodePositionsTrc(n).csv* files - these are the node positions of nth time instance. These files are equivalent to *nodes.csv* file in *Input* folder.

3. ***RoomCoordinates*** - This folder consists of RoomCoordinates.csv file which has coordinates of the geometry defined in the CAD file. The format for *RoomCoordinates.csv* is described by a 2D array. Each row in array represents a triangle as described below.

$$x1, y1, z1, x2, y2, z2, x3, y3, z3$$

where $x1, y1, z1$ are the x, y, z coordinates of first vertex of triangle; $x2, y2, z2$ are the x, y, z coordinates of second vertex, and $x3, y3, z3$ are the x, y, z coordinates of third vertex.

9 Helper Functions

Raytracer folder consists of thirty one functions in which *Raytracer.m* is the main function. Out of thirty one functions, there are nine functions for CAD data extraction out of which three are dedicated to mobility, two are dedicated to backtracking algorithm, three are dedicated to method of images. There are seventeen helper functions in this

folder. A brief description for each of these functions is given below.

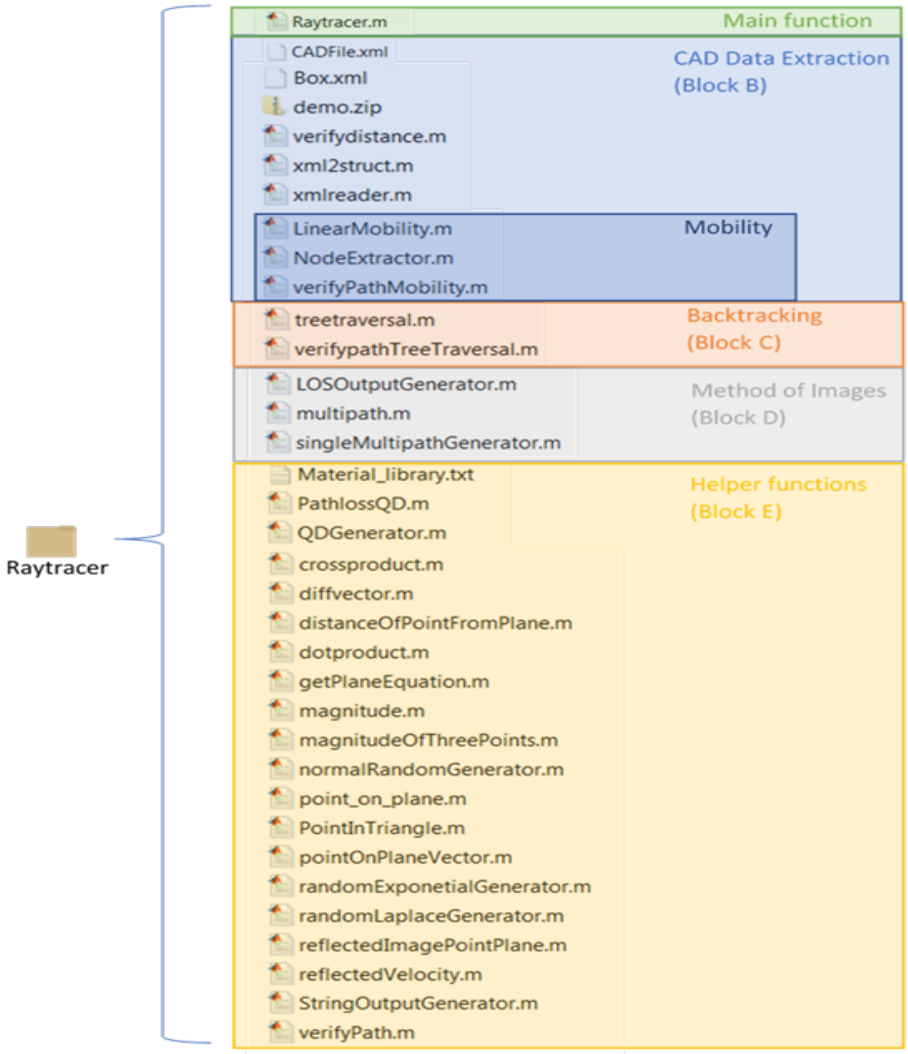

Figure 33: Files present in the *Raytracer* folder.

- ***crossproduct.m*** : This function returns the cross product between two vectors.
- ***diffvector.m*** : This function returns the difference vector between two position vectors (point coordinates).
- ***distanceOfPointFromPlane.m*** : This function returns the normal distance of a point from a plane.

- ***dotproduct.m*** : This function gives the dot product between two vectors.
- ***getPlaneEquation.m*** : This function constructs a plane equation for a given normal and a point on plane.
- ***magnitude.m*** : This function returns the magnitude of a vector.
- ***magnitudeOfThreePoints.m*** : This function returns the magnitude of the position vector of a point in three-dimensional space.
- ***normalRandomGenerator.m*** : This function generates a random number using Normal distribution.
- ***point_on_plane.m*** : This function calculates the projection of a point onto the plane, by constructing a vector passing through the point along the normal of the plane.
- ***PointInTriangle.m*** : This function verifies whether the point lies in the triangle or not using the barycentric coordinate system.
- ***pointOnPlaneVector.m*** : This function returns the intersection point of a vector with the plane.
- ***randomExponetialGenerator.m*** : This function generates a random number using Exponential distribution.
- ***randomLaplaceGenerator.m*** : This function generates a random number using Laplace distribution.
- ***reflectedImagePointPlane.m*** : This function returns the image of a point with respect to a plane (assuming the plane is a mirror).

- ***reflectedVelocity.m*** : This function computes the image velocity of a mobile point with respect to a plane.

- ***StringOutputGenerator.m*** : This function generates the string output which is further saved in a text file.

- ***verifyPath.m*** : Given the CAD model and a vector, one can verify if the vector passes through any one of the triangles in the CAD model. While verifying the intersection of line with triangles, we are only considering intersection present inside the triangle and neglecting others present on the perimeter of the triangle.

10 Time Complexity

NIST Q-D channel realization software is a deterministic raytracer combined with a statistical model. If one is interested in simulating multiple nodes or a large scenario, one has to know the time complexity to decide whether a given scenario is feasible to simulate or not. The time complexity of the NIST Q-D channel realization software is given by

$$\mathcal{O}(n) = rp^2 n^{2m+2},$$

where r is the number of time samples (mobile nodes), p is the number of nodes, m is the order of reflections, n is the number of faces. To reduce time complexity, one has to minimize n and m which are the biggest contributors. This can be done by limiting the order of reflections to at most 2 (preferably one). In addition, one can further reduce the time complexity by decreasing the number of triangles. The number of triangles can be decreased by eliminating triangles which lie beyond a certain distance from Tx and Rx. To limit the number of paths in the output, one can limit the paths based

on the path loss. Duplicate paths must be avoided. The nodes which are too far away from each other can also be ignored.

11 Scenarios

There are six scenarios that are predefined for the user to explore the QDSoftware. Each of the scenarios can be easily activated through *main.m* which are also described below.

- ***Indoor1*** : This represents a simple box scenario in which all the nodes are present inside the box. The box in this scenario is defined by '*box.xml*' file, which can be easily generated using FreeCAD 0.18 software. Since this is an indoor scenario, the switch *indoorSwitch* is activated by assigning a value of '1'. Only for *box.xml*, the Q-D software has the capability of generating random location of nodes. The random locations can be generated by activating the switch *switchRandomization* by assigning a value of '1'.

- ***Indoor2*** : This is also an indoor scenario. However, in contrast to *Indoor1*, the nodes are present within the icasohedron (low resolution sphere) given by '*sphere.xml*' file. For this scenario, the *indoorSwitch* is activated and *switchRandomization* is deactivated by assigning the value '1' and '0', respectively.

- ***Outdoor1*** : This scenario is an outdoor location where nodes are present outside the buildings of a city block. The city block is defined by '*cityBlock.xml*' file. Since this is an outdoor scenario, the switch *indoorSwitch* is deactivated using the value '0'. This is a very large scenario with over hundred thousand triangles. Not all triangles are useful for computation, as some triangles are very far from the nodes. These triangles can be eliminated based on the distance from the

nodes. The *referrencePoint* is the center of limiting sphere and [285.27,178.7,4] is set as the x, y, z coordinates of the center, which is close to the first node. *selectPlanesByDist* is the radius of the limiting sphere and is considered as 10.

- Three scenarios correspond to the ones described in [11], i.e, ***L-Room***, ***Denser-Scenario***, and ***SpatialSharing***.

  Note:If few nodes are present within the buildings and few outside the buildings, then we have to set *generalizedScenario* ='1'.

References

[1] T. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[2] "IEEE 802.11tm wireless local area networks." [Online]. Available: http://www.ieee802.org/11/Reports/tgay_update.htm

[3] A. Maltsev, A. Pudeyev, I. Karls, I. Bolotin, G. Morozov, R. Weiler, M. Peter, and W. Keusgen, "Quasi-deterministic approach to mmwave channel modeling in a non-stationary environment," in *2014 IEEE Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 966–971.

[4] C. Lai, R. Sun, C. Gentile, P. B. Papazian, J. Wang, and J. Senic, "Methodology for multipath-component tracking in millimeter-wave channel modeling," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 3, pp. 1826–1836, March 2019.

[5] M. Born and E. Wolf, *Principles of optics*. Cambridge University, 2009.

[6] 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, July 1997.

[7] H. Lipson, "Amf tutorial: The basics (part 1)," vol. 1, pp. 85–87, 06 2014.

[8] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. McGraw-Hill Higher Education, 2008.

[9] M. Born and E. Wolf, *Principles of optics*. Cambridge University, 2009.

[10] L. Figueiras and J. Deulofeu, "Visualising and conjecturing solutions for herons problem," *Proceedings of the CERME 4 international conference*, p. 420427, 2005.

[11] H. Assasa, J. Widmer, T. Ropitault, A. Bodi, and N. Golmie, "High Fidelity Simulation of IEEE 802.11ad in ns-3 Using a Quasi-deterministic Channel Model," in *Workshop on Next-Generation Wireless with ns-3*, ser. WNGW '19, Florence, Italy, 2019.