ECOLE
POLYTECHNIQUE
DE BRUXELLES

Université Libre de Bruxelles

ELEC-H304
Physique des télécommunications

Philippe DE DONCKER

Rapport de projet - 2021
Le Ray tracing

Étudiants :

Paul SERVAIS
Matricule : 000475842
Théo GUIDE
Matricule : 000478231

Table des matières

Table des figures

1 Introduction

Dans le cadre du cours de Physique des Télécommunications de troisième bachelier à l'Ecole Polytechnique de Bruxelles, il a été proposé aux étudiants de réaliser et d'implémenter un logiciel de *ray-tracing*. Ce logiciel devrait permettre de déterminer la puissance reçue et le débit binaire reçu en chaque point d'un étage de building, pour y définir la qualité des communications sans fil. La forme de cet étage est énoncée comme une donnée du problème.

Ce logiciel fonctionne comme suit : un ensemble d'objets "murs" (de conductivité, de permittivité et d'épaisseur déterminées) est créé et disposé sur une carte correspondant à l'étage voulu. Un émetteur d'ondes électromagnétiques à haute fréquence (une antenne 5G fonctionnant à 27 GHz) est ensuite positionné sur cette carte, et le logiciel balaye toutes les positions potentielles de l'étage. En chacun de ces points de réception est calculé la valeur du champ reçu, et ce champ d'arrivée est considéré comme étant le résultat potentiel d'une incidence directe, de transmissions, de réflexions (simple, double ou triple) ou tous ces modes de transport additionnés selon le principe de superposition.

L'objectif de ce rapport est de fournir une description des résultats obtenus pour un logiciel de *ray-tracing* codé en Python (sur l'interface Spyder), langage de programmation orienté objet. Ce logiciel permet d'obtenir des résultats probants pour l'environnement considéré : le rez-de-chaussée du Metropolitan Museum de New-York. Les prochaines sections concernent la description des fonctionnalités du code, différentes représentations graphiques et validations du code, entre la théorie et la pratique. L'entièreté du code est référencée en annexe.

2 Hypothèses et modélisation physique

Pour approcher la solution réelle de distribution d'intensité du champ dans l'immeuble, plusieurs hypothèses ont été faites pour permettre la modélisation des phénomènes étudiés. L'objectif de ce chapitre est de définir ces hypothèses, et de présenter les modélisations physiques qui en découlent. Ces différentes méthodes permettront ensuite de proposer des vérifications entre la théorie et le code.

2.1 Hypothèses

On considérera un cas à deux dimensions, ce qui implique dans le cas réel qu'émetteurs et récepteurs sont à la même altitude. La diffraction est négligée. Le calcul de la puissance moyenne s'effectue au centre de carrés d'un 1 m^2. Les antennes émettrices sont considérées comme des dipôles idéaux $\frac{\lambda}{2}$, de propriétés suivantes [2] :

1. Résistance d'antenne : $R_a = R_a + R_{al} = 73\Omega$
2. Hauteur équivalente en deux dimensions : $h_e = -\frac{\lambda}{\pi}$
3. Puissance d'émission : $P = 20$dBm

4. Gain - Directivité maximale : $G = D(\theta = 90°) = \dfrac{16}{3\pi} \approx 1,7$

Par ailleurs, on considère qu'il existe une dépendance linéaire entre la sensibilité et le débit binaire. Pour qu'une communication puisse exister, le récepteur doit recevoir une puissance existant dès lors dans l'intervalle de valeurs compris entre -82dBm et -73dBm, correspondant respectivement aux débits binaires $40\frac{\text{Mb}}{\text{s}}$ et $320\frac{\text{Mb}}{\text{s}}$. Ces deux points de fonctionnement permettent de dresser le lien linéaire suivant :

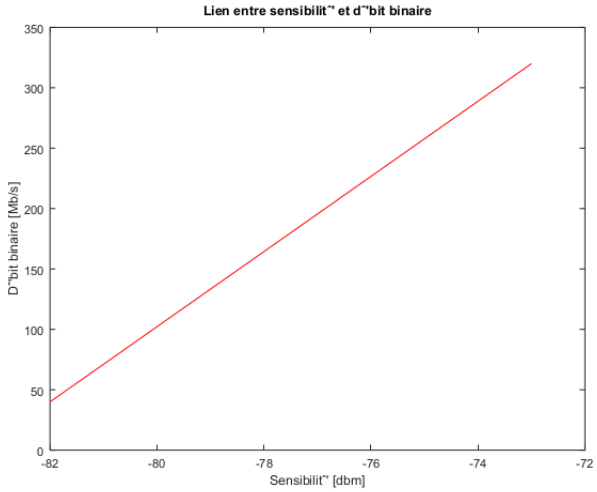


FIGURE 1 – Sensibilité et débit binaire

Les permittivités et conductivités des différents matériaux des murs sont connues. Elles sont représentées dans le tableau suivant.

Matériau	Permittivité	Conductivité
Brique	4.6	0.02
Béton	5	0.014

2.2 Modélisation physique

Un algorithme de *ray-tracing* permet de retracer la trajectoire empruntée par des rayons lors de leur déplacement dans l'espace, et de calculer l'énergie mobilisée lors de ces propagations. Au cours de déplacements entre un point d'émission et un point de réception, un rayon - c'est-à-dire l'onde électromagnétique - peut subir différentes interactions avec les milieux qui composent l'environnement. Dans un milieu donné, le rayon adopte une trajectoire rectiligne. En l'occurrence, le milieu de propagation d'ondes dans le Metropolitan Museum est soit l'air - dont les propriétés sont assimilées à celles du vide dans le cadre de ce projet [1]-, soit les matériaux cités ci-dessus dans le chapitre 2.1.

1. Elon MUSK devra attendre encore un peu avant d'encadrer des peintures dans le vide sidéral.

La puissance au récepteur est déduite de la loi [2] :

$$P_R = \frac{1}{8}\frac{|\underline{V}_{oc}|^2}{R_a} \tag{1}$$

où la tension induite est donnée par :

$$\underline{V}_{oc} = -\frac{1}{\underline{I}_a}\int_D \underline{\vec{J}}(\vec{r}).\underline{\vec{E}}(\vec{r})dV \tag{2}$$

Qui, dans le cas d'une onde plane incidente telle que $\underline{\vec{E}}(\vec{r}) = \vec{E}(\vec{0}).e^{-j\vec{\beta}.\vec{r}}$, peut être réécrite comme la somme :

$$\underline{V}_{oc} = \sum_{n=1}^{N} \vec{h}(\theta_n, \phi_n).\underline{\vec{E}}(\vec{r}) \tag{3}$$

Où, par principe de superposition, la tension induite au récepteur $\underline{V}_{oc}$ est la somme des différentes actions des champs électromagnétiques déduits en ce point. $\vec{h}(\theta_n, \phi_n)$. représente la hauteur de l'antenne, constante d'après les hypothèses précédentes et valant $-\frac{1}{\underline{I}_a}\int_D \underline{\vec{J}}(\vec{r}).e^{-j\vec{\beta}.\vec{r}}dV$. Dès lors, en combinant (1) et (3), on trouve :

$$P_R = \frac{1}{8R_a}\left|\sum_{n=1}^{N} \vec{h}(\theta_n, \phi_n).\underline{\vec{E}}(\vec{r})\right|^2 \tag{4}$$

Et en remplaçant la puissance calculée par la puissance moyenne dans la zone d'$1m^2$ précisée dans les hypothèses, la norme peut entrer dans la somme :

$$\langle P_R \rangle = \frac{1}{8R_a}\sum_{n=1}^{N}\left|\vec{h}(\theta_n, \phi_n).\underline{\vec{E}}(\vec{r})\right|^2 \tag{5}$$

Il convient dès lors de calculer la valeur du champ en chaque point pour obtenir la norme demandée. Cette valeur de champ doit tenir compte des différentes interactions possibles avec l'environnement de propagation. Ces effets sont mentionnés ci-dessous.

2.2.1 Ondes directes

Dans le cas où la trajectoire d'un rayon n'est pas interrompue par un obstacle au cours de sa propagation, on parle d'onde directe (l'onde va être "directement" transmise au récepteur). Dans ce cas, la conservation de l'énergie impose que le champ électro-magnétique transmis au récepteur soit inversement proportionnel à la distance parcourue par le rayon incident entre la source et le récepteur. On a la relation [2] :

$$S = \frac{|\underline{E}|^2}{2Z_0} = \frac{G_E(\theta_n, \phi_n)\ P_E}{4\pi d^2} \tag{6}$$

Où la valeur de $Z_0 = 120\pi$ permet de simplifier pour obtenir :

$$\underline{E}_n = \sqrt{60\ G_E(\theta_n, \phi_n)\ P_E}\ \frac{e^{-j\beta d}}{d} \tag{7}$$

β est le nombre d'onde. D'après les hypothèses, le gain de l'antenne est constant et sa puissance aussi. Seule la valeur de d varie avec la position du récepteur. En utilisant 5, il sera possible de calculer la puissance reçue et le débit binaire auquelle elle est associée (cf. Figure 1).
Dans les cas où l'onde n'est pas directe, celle-ci peut subir des transmissions au travers des murs, ou des réflexions sur les parois (dans le présent projet, ces réflexions peuvent être simples, doubles ou triples avant d'atteindre le récepteur). Les phénomènes de diffraction sont quant à eux négligés par hypothèse. L'objectif des prochaines sections est de modéliser la transmission et la réflexion, qui font apparaître des composantes "multi-trajets" du champ - c'est-à-dire que les ondes transmises et réfléchies se séparent au niveau de l'interface et empruntent deux trajectoires différentes -.

2.2.2 Transmission

La transmission au travers d'un mur est modélisée par un coefficient T_m, multiplicateur de l'expression 15. Ce coefficient intervient lorsque l'onde traverse un mur lors de son trajet entre l'émetteur et le récepteur. La figure 2 propose une illustration de deux phénomènes : à partir d'un champ incident, différents champs sont réfléchis (gauche de l'illustration $\underline{E}_r$), et un champ est transmis (droite de l'illustration $\underline{E}_t$).

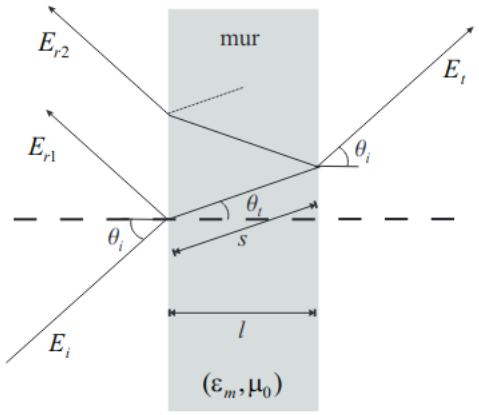


FIGURE 2 – Transmission et réflexion [2]

Si θ_i est l'angle d'incidence de l'onde et θ_t est l'angle de transmission de cette onde, le coefficient est calculé comme suit [2] :

$$T_m(\theta_i) = \frac{(1 - \Gamma_\perp^2(\theta_i))\ e^{-j\beta_m s}}{1 - \Gamma_\perp^2(\theta_i) e^{-2j\beta_m s}\ e^{j\beta 2s\ sin(\theta_i) sin(\theta_t)}} \tag{8}$$

En réalisant par ailleurs l'hypothèse des pertes faibles dans le matériau, on peut réaliser l'approximation : $j\beta \approx \gamma_m$, où γ_m correspond à la constante de propagation dans le matériau. Cette constante est connue et vaut :

$$\gamma_m = j\omega\sqrt{\mu_0(\epsilon - \frac{j\sigma}{\omega})} \tag{9}$$

Par ailleurs, s correspond à la distance parcourue par l'onde dans le mur. Cette distance est $s = \frac{l}{cos(\theta_t)}$. L'angle transmis peut quant à lui être évalué par la méthode de Snell-Descartes :

$$\sin(\theta_t) = \sqrt{\frac{\epsilon_1}{\epsilon_2}}\sin(\theta_i) \tag{10}$$

Dès lors, la seule donnée manquante est $\Gamma_\perp$, le coefficient de réflexion. Celui-ci n'est calculé que dans un cas où la polarisation du champ est perpendiculaire et orientée selon $\vec{1_z}$ (le plan de travail est O_{xy}). L'hypothèse de diélectriques parfaits est aussi imposée, de telle sorte que :

$$\Gamma_\perp = \frac{Z_2\cos(\theta_i) - Z_1\cos(\theta_t)}{Z_2\cos(\theta_i) + Z_1\cos(\theta_t)} \tag{11}$$

Où on peut calculer l'impédance Z d'un matériau comme suit :

$$Z = \sqrt{\frac{\mu_0}{\epsilon - \frac{j\sigma}{\omega}}} \tag{12}$$

Dès lors, l'entièreté des données nécessaires à la formulation analytique de 8 est disponible. Multiplier l'équation 15 par T_m permet d'obtenir la valeur de la composante de champ transmis au travers du mur. Les prochains paragraphes permettront de déterminer les coefficients relatifs aux réflexions simples, doubles et triples.

2.2.3 Réflexion simple

L'ensemble des raisonnements liés aux réflexions sont fondés sur la méthode des images. Cette méthode permet d'établir la course d'un rayon après son interaction avec un mur. Cette situation est représentée en Figure 3.

Si le rayon frappe le mur avec un angle d'incidence θ_i avec la normale, il est possible de déterminer la poursuite du rayon en traçant la droite qui joint le point d'intersection du rayon avec le mur, et le point obtenu par symétrie orthogonale de l'émetteur avec le mur. Un tel point est appelé point image de l'émetteur, et noté I sur le schéma. Cette méthode géométrique permet d'aboutir à un tracé pour la course de l'onde réfléchie. Elle implique de mesurer la distance point (émetteur) - droite (mur) et d'effectuer la symétrie orthogonale des émetteurs pour les différents murs de l'étage considéré.

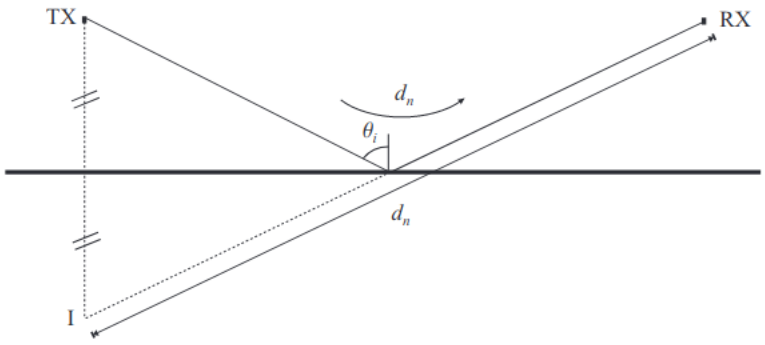


FIGURE 3 – Réflexion simple [2]

Comme dans le cas de la transmission, l'onde réfléchie va subir une atténuation caractérisée par un coefficient de réflexion Γ_m. Ce coefficient donne, si $\underline{E}_r$ est le champ réfléchi et $\underline{E}_i$ est le champ incident :

$$\underline{E}_r = \Gamma_m \cdot \underline{E}_i \tag{13}$$

Il peut être calculé comme suit :

$$\Gamma_m(\theta_i) = \Gamma_\perp(\theta_i) - (1 - \Gamma_\perp^2(\theta_i)) \frac{\Gamma_\perp(\theta_i) e^{-2j\beta_m s} \; e^{j\beta 2s \; sin(\theta_i) sin(\theta_t)}}{1 - \Gamma_\perp^2(\theta_i) e^{-2j\beta_m s} \; e^{j\beta 2s \; sin(\theta_i) sin(\theta_t)}} \tag{14}$$

Où les coefficients représentent chacun la même chose que précédemment. $\Gamma_\perp$ est donné par l'équation 11. Les mêmes approximations pour $j\beta$ pourront être exécutées. L'équation 14 s'obtient en constatant que l'ensemble des champs qui donnent le champs réfléchis sont en progression géométrique, et que cette progression obéit dès lors aux formules associées.

2.2.4 Réflexion double et triple

Les cas de réflexions double et triple exploitent le même raisonnement et la même méthode des images. Comme dans le cas de réflexion simple, le symétrique de la source par rapport au premier mur (indiqué par l'angle d'incidence θ_{i1}) est calculé, et on associe à la réflexion un coefficient Γ_m.

Le prolongement de la droite qui joint ce point I_1 à l'intersection avec le premier mur indique la poursuite du rayon, jusqu'à ce que ce rayon interagisse avec un deuxième mur (selon un angle d'incidence avec la normale θ_{i2}). Comme la source artificielle du rayon était devenue I_1, il convient de trouver le symétrique de ce point par rapport au deuxième mur : I_2. Relier ce point au point d'intersection du rayon avec le deuxième mur permet d'obtenir le prolongement du rayon.

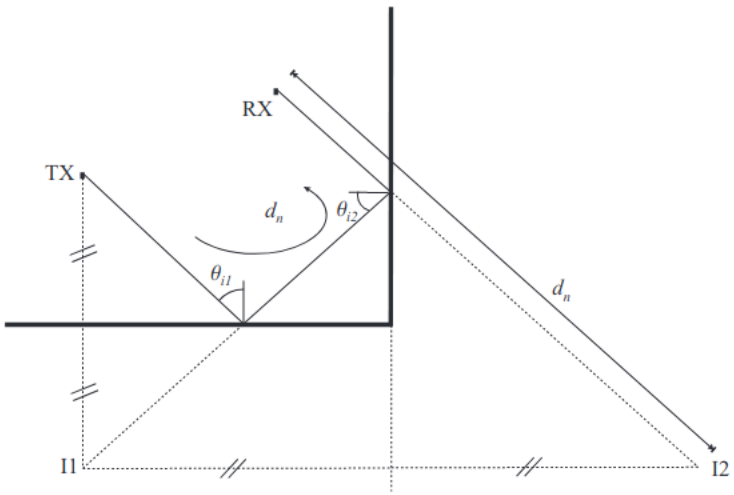


FIGURE 4 – Réflexion double [2]

Ce raisonnement récurrent peut être reproduit pour le cas de réflexion triple : symétriser le point I_2 par rapport au troisième mur, permet d'obtenir I_3. Ce point relié au point d'intersection avec le troisième mur donne le prolongement du rayon réfléchi trois fois. A chaque réflexion est associée un coefficient de réflexion Γ_m calculable d'après l'équation 14. En conclusion, la valeur finale du champ s'exprime :

$$\underline{E}_n = \Gamma_1\Gamma_2...T_1T_2...\sqrt{60\ G_E(\theta_n,\phi_n)\ P_E}\ \frac{e^{-j\beta d_n}}{d_n} \tag{15}$$

L'ensemble des formules de ce chapitre ont été encodées dans un logiciel permettant de définir la valeur de champ en différents points dans le bâtiment, telle que calculable par les formules ci-dessus. L'objectif de la prochaine section est de réaliser la description de ce code et de ses fonctionnalités.

3 Implémentation dans le code

L'ensemble de l'encodage de la modélisation physique des phénomènes décrits a été réalisée à l'aide d'une version 3.8 de Python, langage de programmation orientée objet. Le code a été développé dans l'environnement *Spyder*.

3.1 Choix du langage et de l'environnement de programmation

De premiers essais de code ont été réalisés en GNU - Octave, mais le trop grand nombre de relations entre les différents objets du code ont rapidement rendu meilleur le choix d'un langage de programmation orientée objet. En effet, un tel langage permet de définir des composants tels que les murs, les rayons, les récepteurs, les émetteurs, comme des objets ayant une existence et

des attributs propres. Il est par ailleurs possible de les faire avoir des interactions entre eux. Les réflexions et les transmissions peuvent par exemple être caractérisées comme des interactions entre les murs et les rayons. La caractéristique de compilation dynamique de Python a également facilité le débuggage et assoupli notre utilisation du code.

Le choix de l'environnement de développement de codes à vocation scientifique *Spyder* a été fait pour la qualité d'analyse qu'il offre (notamment la commodité pour tracer des graphiques et des *heatmaps*).

3.2 Architecture du code

Le code est essentiellement composé de deux types de fichiers *.py*. Les premiers incluent des classes, et les seconds incluent des fonctions.

3.2.1 Les classes et objets

Les classes caractérisent des objets, comme le bâtiment dont l'attribut principal est une liste de murs(classe contenue dans *building.py*), l'émetteur, dont les attributs sont la position, le gain, la hauteur (classe contenue dans *emetteur.py*), le récepteur (classe contenue dans *recepteur.py*), ou encore les murs dont les attributs sont une position de départ et de fin, la permittivité et la conductivité (*wall.py*). Chacun de ces objets dispose d'une fonction *.plot()* qui permet de les représenter sur l'interface graphique.

Les relations entre ces différentes classes sont dirigées par des fonctions disponibles dans d'autres fichiers. La fonction *main()* génère la carte en créant un objet *building* par le biais de la fonction *buildfield()*. Une liste d'émetteurs, qui peut n'en contenir qu'un, est également créée et les dimensions de la carte sont définies. Ensuite, la fonction *heatmap_dBm()* qui prend ces différents éléments en argument est lancée. Cette fonction balaye le plan et place un récepteur en chaque point au centre des carrés d' $1m^2$ du plan. En chacun de ces points est calculée la valeur en dBm de la puissance, par le biais de la fonction *.totaldBm()*.

Lorsque par contre n émetteurs sont encodés, la fonction balaye alors le plan n fois et calcule en chaque point la puissance reçue en W via la fonction *.totalW()*. Ces puissances sont à la fin de chaque balayage complet additionnées case par case aux précédentes et enfin après n boucles, converties en dBm.

Les fonctions *.totaldBm()* et *.totalW()* diffèrent donc uniquement l'un de l'autre par l'unité dans laquelle elles retournent la puissance calculée en un point.

Quand toutes ces données sont calculées, la fonction *heatmap_dBm()* trace également une *heatmap*.

3.2.2 Les fonctions : calcul des coefficients, des ondes directes, des transmissions et réflexions

Les fonction *totaldBm()* et *.totalW()* prennent en argument le bâtiment, l'émetteur et le récepteur, et calculent le total de la puissance au point récepteur, par addition des composantes multi-trajet *reflexion0, reflexion1, reflexion2, reflexion3*, qui prennent en compte les réflexions et transmissions de rayons ayant subi i réflexions ($\forall i = 0, 1, 2, 3$). La valeur de puissance associée à ces champs est ensuite calculée d'après 5 et convertie en dBm dans le cas de la fonction *totaldBm()*. L'analyse détaillée de ces fonctions $reflexion_i()$ est l'objectif du prochain point. Comme indiqué précédemment, le calcul des transmissions et des réflexions implique le calcul préalable d'une série de coefficients. Ces coefficients ont été calculés grâce à différentes fonctions, disponibles dans le fichier *coefficients.py*. Ce fichier calcule :

1. *coefftransmission()* : Le coefficient de transmission T_m sur base de l'équation 8
2. *coeffreflexion()* : Le coefficient de réflexion Γ_m sur base de l'équation 14
3. *gammaperp()* : Le coefficient de réflexion perpendiculaire $\Gamma_\perp$ sur base de l'équation 11
4. *thetai()* : L'angle d'incidence θ_i sur base de la méthode des images
5. *thetat()* : L'angle transmis θ_t calculé par la loi de Snell-Descartes 10

Dès lors, pour chaque doublet de positions émetteur - récepteur, et à partir de la liste des murs (le bâtiment), les fichiers $reflexion_i(\forall i = 0, ..., 3)$ permettent les calculs de transmissions et réflexions de champs. La fonction *reflexion0()* correspond au cas sans réflexion, avec des éventuelles transmissions,le cas d'onde direct n'est que le cas particulier de cette onde lorsqu'il n'y a pas de transmission. La fonction *reflexion1()* prend en compte une seule réflexion et les transmissions durant tout le parcours du rayon, et ainsi de suite.

Pour chaque mur du bâtiment, la fonction *reflexion1()* indique le symétrique de l'émetteur (nommé *mirror*) par rapport à ce mur. Si il y a une intersection entre le mur et la droite qui joint le récepteur au symétrique, alors le code calcule le point d'intersection entre les deux, et identifie deux coefficients de transmissions : l'un entre l'émetteur et le point d'intersection, l'autre entre le point d'intersection et le récepteur. Le coefficient de réflexion est ensuite calculé, ainsi que la distance parcourue par le rayon. En multipliant le champ par tous les coefficients calculés et en le divisant par la distance obtenue, on obtient par 15 le total de l'interaction avec un mur, ce que cette fonction renvoie.

La fonction *reflexion2()* fonctionne de la même manière, si ce n'est que l'examen se fait deux fois pour chaque mur du plan : le symétrique I_1 est créé, et sert à créer le symétrique I_2, dénommé *mirror2* dans le code. Ce point permet de réaliser la même opération que pour la réflexion unique, si ce n'est que le coefficient total devient dans l'équation 15, par application récursive du même raisonnement :

$$(\Gamma T)_{tot} = T_1 \; T_2 \; ... \; \Gamma_1 \; \Gamma_2 \; ... \tag{16}$$

La fonction *reflexion3()* fonctionne de manière analogue.

4 Vérification du code

Cette section concerne la validation du code par l'exploitation de résolutions analytiques pour chacun des phénomènes décrits ci-dessus. Pour chaque interaction avec les obstacles du plan, une situation a été mise en scène de telle sorte que les calculs analytiques et informatiques (disponibles en annexe) puissent être comparés. En effet, les calculs prennent énormément de temps dans le cas où la géométrie de l'environnement est complexe, comme c'est le cas pour la situation finale étudiée.

4.1 Onde directe

La première situation étudiée est celle de l'onde directe, pour un émetteur placé en (100,65) et un récepteur placé en (150,30). Les dimensions du plan sont prises égales à celles du bâtiment, et tous les obstacles sont enlevés.

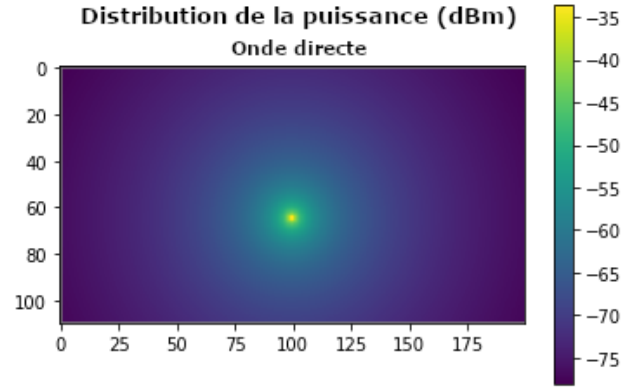


FIGURE 5 – Heatmap pour l'onde directe

Pour rappel, le champ au récepteur d'une onde directe est donné par 7, ce qui implique de calculer préalablement G_E, P_E, β, d.Le gain et la puissance G_E, P_E ont été identifiés dans les hypothèses du rapport. Il convient également de définir la valeur de la longueur d'onde, pour une antenne fonctionnant à la fréquence de 27 GHz. Ces constantes peuvent être calculées comme suit pour une antenne $\frac{\lambda}{2}$:

1. $\beta = \dfrac{2\pi}{\lambda} = \dfrac{\omega}{c} = 565,88 \ \dfrac{rad}{m}$
2. $\omega = 2\pi f = 1,6965.10^{11} \ \frac{rad}{s}$
3. $d = \sqrt{(100,5 - 150,5)^2 + (65,5 - 30,5)^2} = 61,033 \ m$
4. $\lambda = \dfrac{c}{f} = 0.011103 \ m$

Et le calcul du champ par 7 donne :

$$\underline{E} = 8,9336.10^{-4} + 5,2331.10^{-2}i \ \frac{N}{C} \tag{17}$$

Et pour P :

$$P_R = 5,857.10^{-11}\ W \tag{18}$$

ou

$$P_R = -72,323\ dBm \tag{19}$$

Ce résultat est le même que celui obtenu par le logiciel, qui aboutit à une puissance identique 25.

4.2 Transmission

Pour la transmission, on choisit un émetteur placé en (100,65) et un récepteur placé en (150,30). Entre les deux, en y = 45, on place un mur qui traverse l'entièreté du plan. Cette situation est représentée sur la figure suivante :

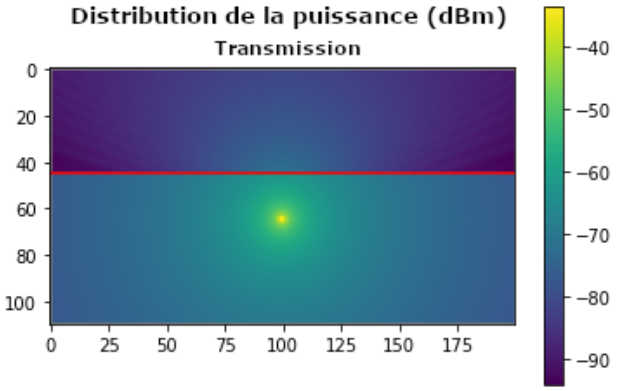


FIGURE 6 – Heatmap pour la transmission

on ne s'intéresse qu'à la transmission de l'onde au travers du mur. En appliquant l'hypothèse des pertes faibles, on peut déduire de 9 et de 12 les valeurs :

Matériau	Permittivité ϵ_r	Conductivité σ	Propagation γ_m	Impédance Z
Brique	4.6	0.02	1,7569 + 1213,3877i	175.69242 + 0.25439i
Béton	5	0.014	1,1796 + 1265,0433i	168,51856 + 0,15714i

Et on peut assimiler le terme $j\beta$ à γ_m. On peut en outre calculer l'angle θ_i et trouver d'après la loi de Snell-Descartes que, dans le cas d'un mur en brique et pour ces positions d'émission et de réception :

$$\theta_t = \arcsin(\sqrt{\frac{1}{\epsilon_r}}\sin(\theta_i)) = -0,39193\ rad \tag{20}$$

Qui permet de calculer le coefficient de réflexion perpendiculaire $\Gamma_\perp$ et le coefficient de transmission :

$$T_m(\theta_i) = -0,2572357 + 0,0079074i \tag{21}$$

D'où découle le champ électromagnétique :

$$\underline{E_n} = 0,0132761 + 0,0013912i \ \frac{N}{C} \tag{22}$$

Et la puissance au récepteur :

$$P = 3.8114.10^{-12} \ W \tag{23}$$

Qui donne en dBm :

$$P = -84.189 \ dBm \tag{24}$$

Les résultats obtenus sont concordants avec ceux du logiciel, qui obtient $P_R = -84.1839$ dBm 26.

4.3 Réflexion simple

La réflexion simple a été testée en fixant un émetteur en $(100, 65)$, le récepteur en $(150, 55)$, et un mur en $y = 45$. Les prochains calculs s'intéressent à la réflexion simple seulement. En appliquant la méthode du point image, on trouve le point image $I_1 = (100, 25)$. La droite qui le joint au récepteur donne la valeur $(\frac{400}{3}, 45)$ comme point d'intersection avec le mur.

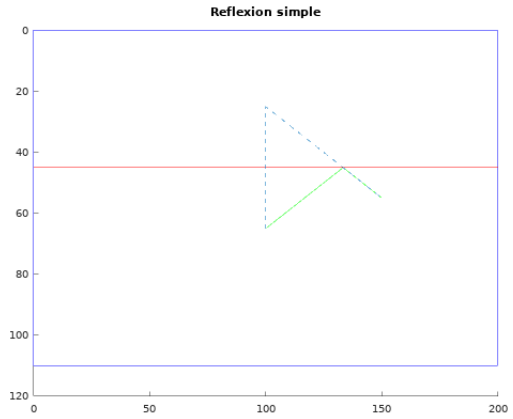


FIGURE 7 – Schéma de la résolution

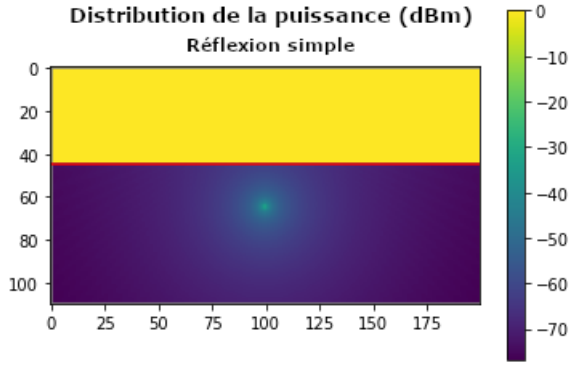


FIGURE 8 – Heatmap pour la réflexion simple uniquement

Ce point d'intersection permet de mesure l'angle θ_i comme :

$$\theta_i = arctan\left(\frac{\frac{400}{3} - 100}{45 - 65}\right) = -1,0304 \ rad \tag{25}$$

Qui, par la loi de Snell-Descartes, donne :

$$\theta_t = -0,41131 \ rad \tag{26}$$

Ces angles permettent de calculer $\Gamma_\perp$ et le coefficient de réflexion :

$$\Gamma_m = -0,625002 - 0,037039i \tag{27}$$

Qui permet d'obtenir le champ dû à la réflexion simple :

$$\underline{E_n} = 0,0340304 + 0,0042339i \ \frac{N}{C} \tag{28}$$

Et les puissances :

$$P_R = 2,5154.10^{-11} \ W \ ou \ P_R = -75,994 \ dBm \tag{29}$$

Toutes ces valeurs concordent avec celles calculées par le logiciel 3.

4.4 Réflexion double et triple

4.4.1 Réflexion double

Pour calculer la réflexion double, il faut considérer que le rayon est réfléchi deux fois. La situation choisie est la suivante : un émetteur est placé en $(100, 65)$, et un récepteur est placé en $(130, 60)$. Une telle situation est schématiquement en figure 9. La Figure 10 représente la distribution totale de la puissance due - uniquement - à la réflexion double avec une même configuration d'émetteur.

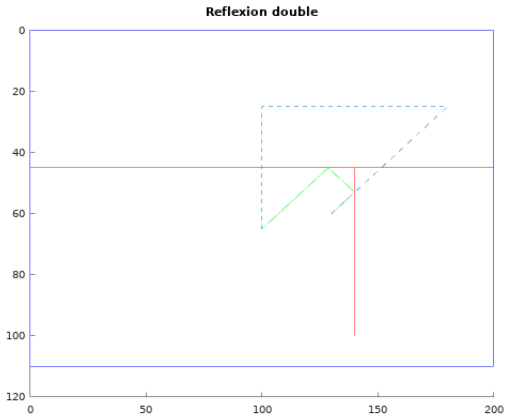


FIGURE 9 – Schéma de la résolution

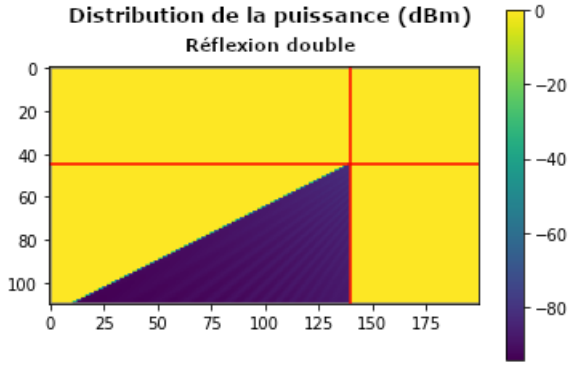


FIGURE 10 – Heatmap pour la réflexion double uniquement

La méthode des images a été appliquée, et les différents prolongements et symétries sont représentés sur la figure 9. En choisissant comme points images : $I_1 = (100, 25)$; $I_2 = (180, 25)$, on trouve les valeurs d'intersections : $(\frac{900}{7}, 45)$ et $(140, 53)$.
Ces points permettent de calculer les valeurs de $\theta_i(n)_{\forall n=1,2}$ et de $\theta_t(n)_{\forall n=1,2}$

$$\theta_{i,1} = arctan\left(\frac{\frac{900}{7} - 100}{45 - 65}\right) = -0.96007 \ rad \quad et \quad \theta_{i,2} = arctan\left(\frac{53 - 45}{140 - \frac{900}{7}}\right) = 0.61073 \ rad \tag{30}$$

Par application de Snell-Descartes, on trouve par ailleurs que :

$$\theta_{t,1} = -0,39193 \ rad \quad et \quad \theta_{t,2} = 0,27067 \ rad \tag{31}$$

Desquels découlent $\Gamma_\perp$ et les coefficients de réflexion :

$$\Gamma_{m,1} = -0,4968224 - 0,0047235i \quad et \quad \Gamma_{m,2} = -0,471342 + 0,043682i \tag{32}$$

A partir desquels on trouve le coefficient total de réflexion $\Gamma_m = \Gamma_{m,1}.\Gamma_{m,2}$, et le champ électrique est :

$$\underline{E} = 0,0012284 + 0,0122455i \ \frac{N}{C} \tag{33}$$

Et les puissances :

$$P_R = 3,2397.10^{-12} \ W \quad ou \quad P_R = -84,895 \ dBm \tag{34}$$

Toutes ces conclusions sont concordantes avec celles du logiciel, qui trouve une puissance finale de $P_R = -84,90$ dBm 28.

4.4.2 Réflexion triple

Le même raisonnement a été appliqué pour valider le code lié à la réflexion triple. Un mur de briques a été ajouté en $y = 80$, un émetteur a été placé en $(100, 65)$, le récepteur a quant à lui été placé en $(95, 70)$. Cette situation est représentée par les deux figures ci-dessous :

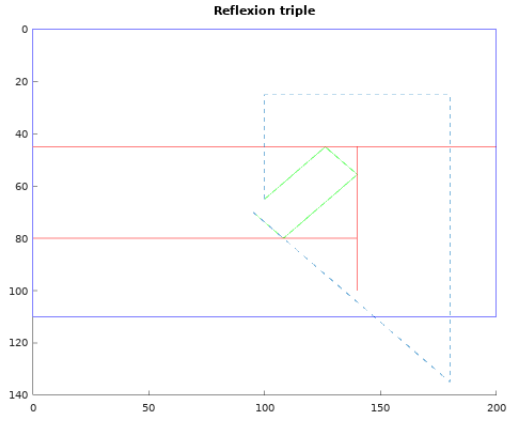


FIGURE 11 – Schéma de la résolution

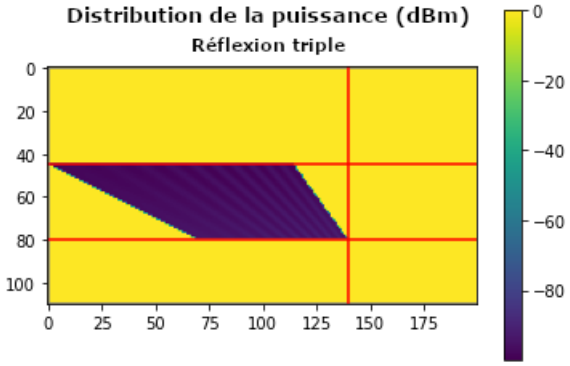


FIGURE 12 – Heatmap pour la réflexion triple uniquement

La méthode des images permet de trouver les points d'intersection avec les murs $(108, 08; 80)$,$(140; 55, 59)$ et $(126, 15; 45)$ desquels découlent les angles incidents $\theta_{i,n\ \forall n=1,2,3}$ et $\theta_{t,n\ \forall n=1,2,3}$. Ils permettent à leur tour de calculer $\Gamma_{m,n\ \forall n=1,2,3}$ et Γ_m, pour finalement trouver :

$$\underline{E} = -0,0021975 + 0,0021780i \ \frac{N}{C} \tag{35}$$

Et les puissances :

$$P_R = 2,0475.10^{-13} \ W \quad ou \quad P_R = -96.888 \ dBm \tag{36}$$

A nouveau, ces valeurs tendent vers celles calculées par le logiciel (qui trouve $P_R = 2,0473.10^{-13}$ W).

L'ensemble des différentes interactions possibles a donc été vérifié, et les valeurs calculées par le code sont identiques aux valeurs mesurées analytiquement, par résolution manuelle (en utilisant la méthode des images, notamment). L'addition de toutes les valeurs de champ trouvées ci-dessus permet, pour un point de réception donné, de définir la puissance totale, de laquelle découle le débit binaire auquel la puissance est liée linéairement. La concordance du logiciel et de la méthode analytique se fait à 3 chiffres significatifs près, de petites erreurs d'arrondi s'accumulant entre les différentes méthodes.

5 Application au Metropolitan Museum de New York

Le chapitre précédent a montré que les différentes interactions entre les objets du plan fonctionnaient pour le logiciel. Le code est dès lors applicable à des situations plus complexes comme celle du Metropolitan Museum de New York (MET). Le temps de calcul associé à la mesure de la distribution de puissance et de débit binaire dans le MET est dépendant du nombre de réflexions dont le code tient compte. Le logiciel prend quelques minutes à s'exécuter si l'on ne tient compte que de la réflexion simple, mais peut prendre jusqu'à 18 heures de temps d'exécution lorsque l'on tient compte de tous les types de réflexion. Le prochain chapitre présente le cas à un émetteur, et le cas optimisé à plusieurs émetteurs.

5.1 Cas à un seul émetteur

Le premier cas simulé est celui d'un émetteur placé en $(100; 65)$. Cet émetteur est placé sur la carte générée par la fonction *.buildfield()*. Le cas présenté ci-dessous intègre tous les types de réflexions considérés dans le cadre de ce projet (simple, double et triple).

Un des objectifs de la représentation de la sensibilité sur une *heatmap* est d'associer aux puissances des valeurs numériques de débits binaires. La communication n'est possible qu'entre les valeurs de $40\frac{\text{Mb}}{\text{s}}$ et $320\frac{\text{Mb}}{\text{s}}$. Aussi, ces valeurs extrêmes sont choisies pour la représentation.

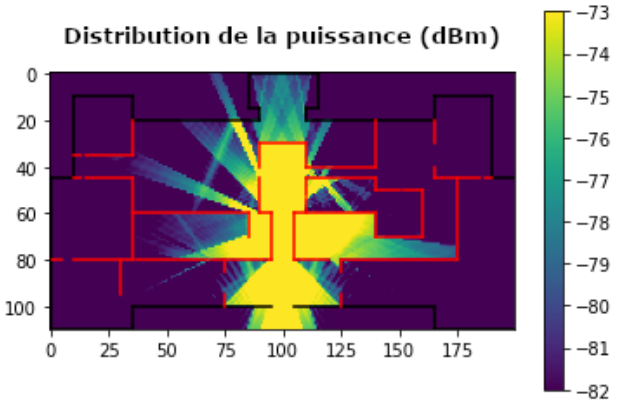


FIGURE 13 – Distribution de la puissance

En exploitant le lien linéaire entre la sensibilité et le débit binaire tel que présenté dans l'introduction, on trouve :

$$DB_{\frac{Mb}{s}} - 40 = \frac{320 - 40}{-73 + 82} \left(P_{dBm} + 82\right) \tag{37}$$

Ce qui permet d'associer à chaque valeur de puissance, une valeur de débit binaire, et d'obtenir la figure suivante :

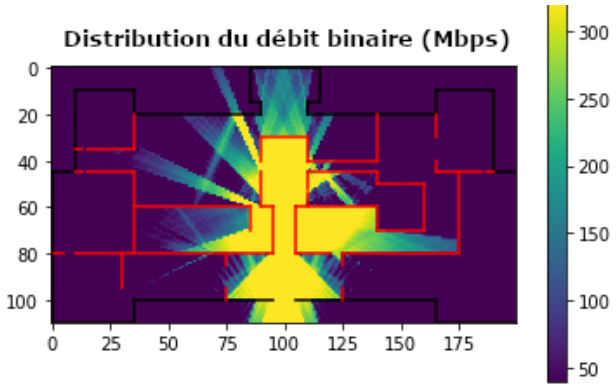


FIGURE 14 – Distribution du débit binaire

Cette figure est exploitable pour définir les différentes zones où la communication est possible dans le bâtiment. En effet, on distingue trois différents types de zones :

1. Zone 1 : jaune claire. Cette zone est de forte intensité ($> 300\frac{\text{Mb}}{\text{s}}$) et les communications peuvent s'y faire sans problème.
2. Zone 2 : entre le bleu foncé et le vert. Cette zone est d'intensité intermédiaire et caractérise les zones où le débit figure entre $40\ \frac{Mb}{s}$ et$300\frac{\text{Mb}}{\text{s}}$. La communication peut toujours se faire dans cette zone
3. En deça de $40\frac{\text{Mb}}{\text{s}}$, la communication est impossible. Cette zone est caractérisée par le violet, et occupe quasiment tout le plan.

Ces représentations permettent de montrer que la qualité du débit binaire est dépendante de deux choses.

D'une part, de la proximité du récepteur et de l'émetteur, même quand ils sont séparés par un mur. Cette sensibilité est particulièrement mise en évidence par les deux zones de transmission à gauche et à droite du récepteur, où l'intensité est particulièrement importante malgré la présence d'un mur.

D'autre part le débit binaire est dépendant du mode d'incidence de l'onde sur le récepteur. On observe par exemple que l'intensité est grande dans les endroits jusqu'où l'onde a pu se propager directement ou par une réflexion simple. C'est le cas de tout le couloir vertical environ compris entre $x = 95$ et $x = 110$. Ce résultat n'est pas surprenant, étant donné l'ordre de grandeur des coefficients de réflexion calculés ci-dessus (qui, multipliés par le champ et élevés au carré, peuvent conduire à de très grandes divisions de l'onde directe).

Ce deuxième critère est important dans l'implication qu'il a sur le choix d'installation des émetteurs. En effet, étant donné la forte sensibilité de la propagation à la présence d'obstacles, les

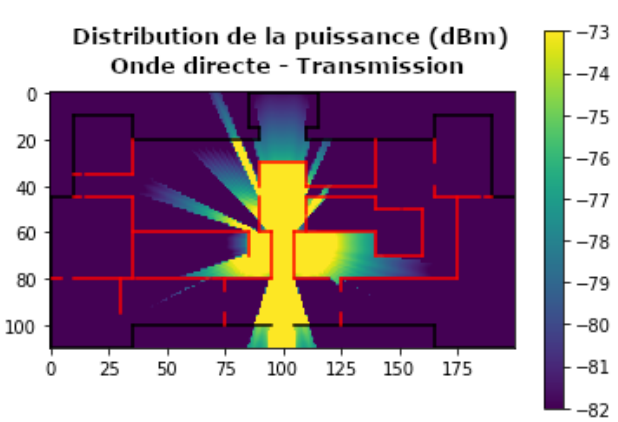


FIGURE 15 – Ondes directes et transmission

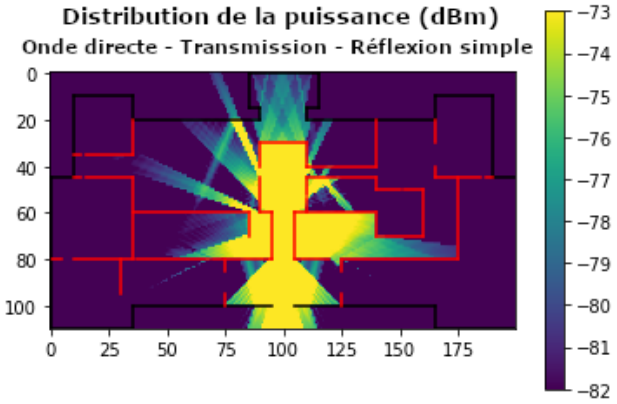


FIGURE 16 – Ondes directes, transmission et réflexion simple

émetteurs ont tout intérêt à être placés dans les zones les moins "closes" possibles. La géométrie de l'environnement du MET ne joue pas en faveur d'un déploiement économique d'émetteurs, étant donné la présence de plusieurs renfoncements et pièces quasiment fermées (ou présentant des embrasures de portes de 2m de large). La prochaine section poursuit cette discussion sur le positionnement des émetteurs. La figure 17 présente la situation sans tenir compte des réflexions, et la figure 16 inclut également la réflexion simple. La comparaison avec le résultat total représenté en figure 14 montre que les effets d'ondes directes, de transmissions et de réflexions simple sont dominants.

5.2 Cas à plusieurs émetteurs

Compte tenu la distribution du débit binaire établie pour un seul émetteur ci-dessus, il a été entrepris de définir les positions d'un nombre minimal de stations de base, telles que ces stations de base permettent la couverture optimale de l'étage considéré. Il serait possible de réaliser un algorithme de force brute réalisant les distributions du débit binaire pour un émetteur se déplaçant sur toutes les cases du plan. Cependant, le temps de calcul propre au logiciel de *ray-tracing* rend cette approche naïve impraticable, si on intègre la réflexion double et la réflexion triple au problème.

Dans une première visualisation, ces deux types de réflexion seront négligées (sans, comme cela a été indiqué au point précédent, perdre de trop grosses informations sur les tendances que prennent les distributions de débit binaire). Deux critères sont également posés :

1. La communication doit être possible partout dans le musée
2. Le nombre d'antennes doit être minimisé

5.2.1 Arbre de visibilité [3] [1]

Une approche intuitive et une analyse rapide des figures ci-dessus conduisent à la conclusion que les obstacles affaiblissent considérablement la propagation des ondes : les émetteurs ont dès lors tout intérêt à être positionnées dans des endroits où les rayons ne sont pas interrompus. Les embrasures de portes et de couloir sont par exemple des endroits critiques, parce qu'elles permettent "d'éclairer" l'entièreté de deux pièces. On peut représenter le système "émetteur"-pièce comme une lampe éclairant un plan et générant là des zones éclairées, ici des zones d'ombre. Pour illustrer cette différence d'émission, les "arbres de visibilité" de deux situations ont été représentées. Il apparaît clairement par construction géométrique que les embrasures permettent d'éclairer mieux la totalité des lieux(les zones éclairées sont en rose, les zones non éclairées en blanc, les murs sont en gris et la source en rouge). Dès lors, la construction d'un arbre de visibilité

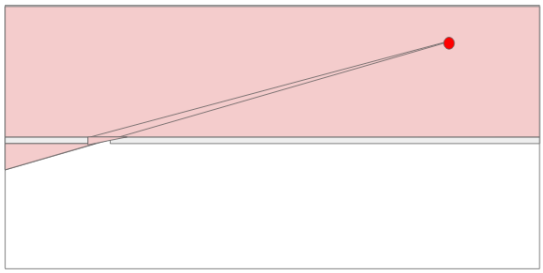


FIGURE 17 – Zone éclairée par une source enfermée

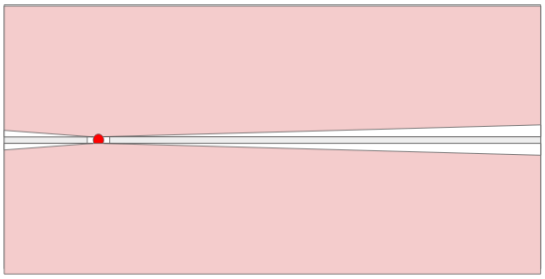
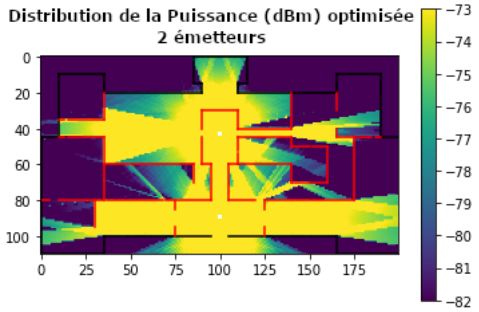


FIGURE 18 – Zone éclairée par une source dans l'embrasure d'une porte

est un outil utile pour exclure les points enfermés des potentiels lieux d'émission. Dès lors, il est possible, en fixant le nombre d'émetteurs, de réaliser plusieurs distributions de débit binaire et de les comparer.

5.2.2 Double émission

En appliquant ce raisonnement, on obtient que des positions potentiellement optimales pour deux émetteurs sont les suivantes : en $(100, 42)$ $(100, 90)$. Etant donné la lourdeur des calculs, seule la réflexion simple a été prise en compte. Voici la figure obtenue :



On constate à nouveau que les différentes zones sont plus ou moins bien éclairées, selon leur proximité avec les émetteurs, et la présence de murs entre la source et le récepteur.

5.2.3 Triple émission

A nouveau, on place des émetteurs en $(100, 90)$, $(14, 40)$ et $(186, 45)$, et on obtient la figure 19 : Qui couvre une plus large zone que la Figure 5.2.2. Un raisonnement similaire donne les

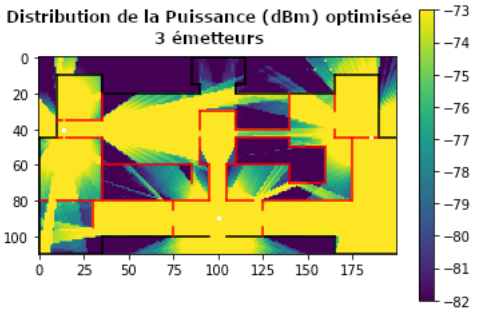


FIGURE 19 – Distribution de la puissance - triple émission

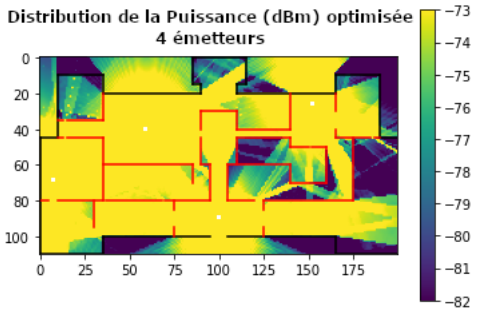


FIGURE 20 – Distribution de la puissance - quadruple émission

positions $(60, 40)$ $(100, 90)$ $(151, 25)$ $(8, 70)$ pour 4 émetteurs 20. Il convient dès lors d'ajouter un émetteur dans la partie droite du bâtiment.

5.2.4 Quintuple et sextuple émissions

On place des émetteurs en $(60, 40)$, $(100, 90)$ $(151, 25)$ $(8, 70)$ et $(150, 53)$. Cette couverture

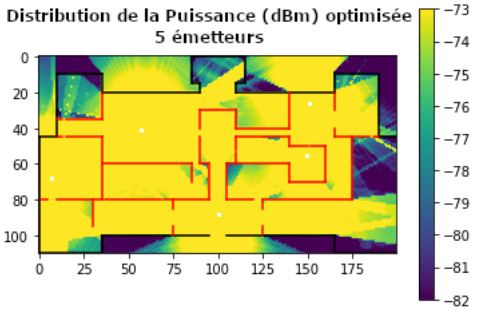


FIGURE 21 – Distribution de la puissance - quintuple émission

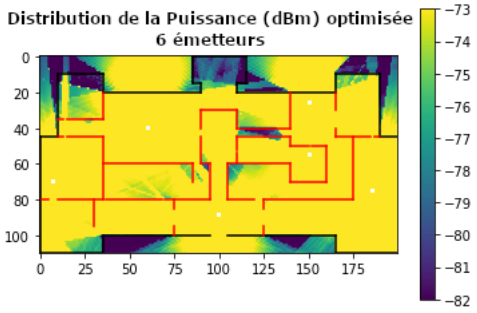


FIGURE 22 – Distribution de la puissance - sextuple émission

du MET devient raisonnable, si ce n'est la zone en bas à droite du plan, à laquelle on ajoute un dernier émetteur pour obtenir la figure 24. On ajoute cet émetteur en $(186, 70)$.
La distribution obtenue pour six émetteurs est relativement satisfaisante : elle permet de communiquer partout dans le musée, si ce n'est très au Nord de celui-ci, dans l'enclave gauche et

centrale. Il est possible que l'implication des réflexions doubles et triples permettent encore de réduire ce défaut.

6 Conclusion

Ce rapport a montré quelles étaient les hypothèses et modélisations requises dans la formation d'un logiciel de *ray-tracing*. Ce logiciel a par ailleurs été codé dans un langage de programmation orientée objet, Python, et dans un environnement de programmation scientifique, *Spyder*.
Ce code a pu être vérifié sur base de développements théoriques élémentaires, et ensuite a pu être étendu à plusieurs émissions par principe de superposition. Ces différentes fonctionnalités ont permis d'établir les distributions de puissance et de débit binaire dans le Metropolitan Museum de New York (MET) pour différentes positions d'émissions, et plusieurs quantités d'émetteurs.

Pour aboutir à une couverture quasi totale du musée, 6 émetteurs à haute fréquence (27GHz) ont dû être placés. Les ondes produites par ces antennes 5G subissent de fortes atténuations au cours de leur propagation.

Il est dès lors raisonnable, dans une volonté d'économie d'argent, de maintenance et de matériaux rares, d'interroger la nécessité pour le Metropolitan Museum de New York d'investir dans un tel déploiement d'infrastructures 5G, sachant d'une part qu'il s'agit d'un musée de peintures et non d'un salon de la voiture autonome et que d'autre part, des bornes WI-FI fonctionnant à 2.45 GHz produisent plus largement les gammes de débits binaires souhaitées (entre 40 $\frac{Mb}{s}$ et 320 $\frac{Mb}{s}$), le tout dans un moindre coût économique et écologique.
Pour illustrer cette différence, les figures suivantes présentent deux cas : à gauche est représentée la couverture du MET par une antenne 5G fonctionnant à 27GHz, à droite est représentée la couverture du MET par une borne Wifi fonctionnant à 2.45 GHz. Les deux antennes sont situées en (100 ;65). Seules les ondes directes, transmissions et réflexions simples ont été prises en compte.

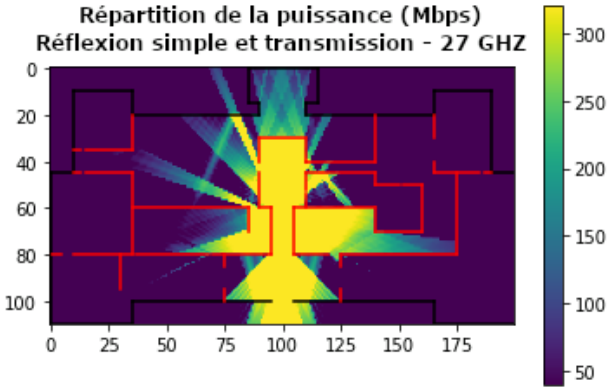


FIGURE 23 – Antenne 5G à 27GHz

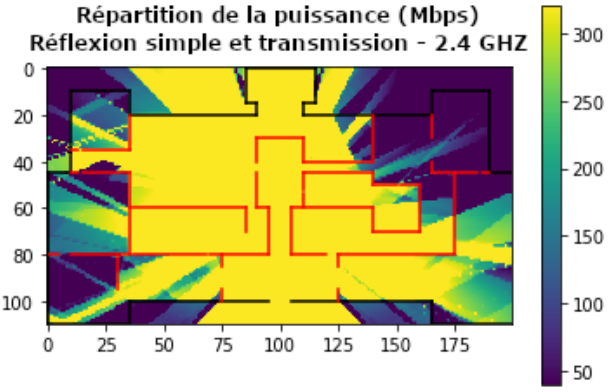


FIGURE 24 – Borne Wifi à 2.45 GHz

Bibliographie

[1] Lilian AVENEAU. "Modèle, calculs et applications de la visibilité en dimension n". In : *Université de Poitiers* (2013). DOI : tel-00979543.

[2] Philippe DE DONCKER. *Physique des Télécommunications*. Presses Universitaires de Bruxelles. ULB, 2021.

[3] Katia JAFFRÈS-RUNSER. "Méthodologies pour la planification de réseaux locaux sans-fil". In : *INSA de Lyon* (2005). DOI : tel-00406342.

A Validation du code - comparaison

A.1 Onde directe

```
d = 61.032778078668514

En = (0.0008912943857540021+0.052320747585207 26j) N/C
P = 5.856967095544728e-11 W
P = -72.32327215832633 dBm
```

FIGURE 25 – Résultats du code - Onde directe

A.2 Transmission

```
Point d'intersection : (128.57142857142858 , 45)
s = 0.5410229589945397
theta t = 0.3919257497796166
theta i = 0.9600703624056882
gammam : (1.7565123037446524+1213.6747027928786j)
Coefficient Gamma perpendiculaire = (-0.5512116470352382+0.0005037689276795002j)
Coefficient de transmission : (-0.2574144711150352+0.007161414851355376j)

d = 61.57380103766305

En = (0.013279974895122103+0.0014312786787889 33j) N/C
P = 3.816006063437187e-12 W
P = -84.18390943895818 dBm
```

FIGURE 26 – Résultats du code - Transmission

A.3 Réflexion simple

```
Point de réflexion : (133.33333333333334 , 45)
s = 0.5454949018786949
theta i = 1.0303768265243125
theta t = 0.4113075171649741
gammam : (1.7565123037446524+1213.6747027928786j)
Coefficient Gamma perpendiculaire = (-0.5851487380042628+0.0004758621758234 6854j)
Coefficient de réflexion : (-0.6410623015613548+0.020658831588278807j)

d = 58.309518948453004
Coefficient de réflexion totale = (0.4105340871605485-0.02648719625110086j)

En = (-0.022532629804294053-4.3543779220546535e-06j) N/C
P = 1.0859821049243616e-11 W
P = -79.64177331097663 dBm
```

FIGURE 27 – Résultats du code - Réflexion simple

A.4 Réflexion double

```
Point de réflexion 1 :(128.57142857142858 , 45)
s = 0.5410229589945397
theta i = 0.9600703624056882
theta t = 0.3919257497796166
gammam : (1.7565123037446524+1213.6747027928786j)
Coefficient Gamma perpendiculaire = (-0.5512116470352384+0.0005037689276795002j)
Coefficient de réflexion : (-0.4965824826600942-0.0047234191363349064j)

Point de réflexion 2 :(140 , 53.0)
s = 0.5188919926755987
theta i = 0.6107259643892091
theta t = 0.27067113477361077
gammam : (1.7565123037446524+1213.6747027928786j)
Coefficient Gamma perpendiculaire = (-0.4322542163120504+0.0005884272042319595j)
Coefficient de réflexion : (-0.471047600423962+0.04366482658623134j)

d = 61.032778078668514
Coefficient de réflexion total = (0.2341202341470932-0.019458232741146064j)

En = (0.0012267393339904 35+0.012232002661800807j) N/C
P = 3.2325132544349884e-12 W
P = -84.90459685684638 dBm
```

FIGURE 28 – Résultats du code - Réflexion double

A.5 Réflexion triple

```
Point de réflexion 1 :(126.15384615384615 , 45)
s = 0.5382804535070839
theta i = 0.9179496956941222
theta t = 0.3794086614105083
gammam : (1.7565123037446524+1213.6747027928786j)
Coefficient Gamma perpendiculaire = (-0.532674827074587+0.0005183080305073684j)
Coefficient de réflexion : (-0.4827445290353367 7+0.024638996445287106j)

Point de réflexion 2 :(140 , 55.58823529411765)
s = 0.5213473656333195
theta i = 0.6528466311007742
theta t = 0.28715521409933864
gammam : (1.7565123037446524+1213.6747027928786j)
Coefficient Gamma perpendiculaire = (-0.4428107730280669 3+0.0005817425164654019j)
Coefficient de réflexion : (-0.4399072488060427+0.05745979397865025j)

Point de réflexion 3 :(108.07692307692308 , 80)
s = 0.5382804535070839
theta i = 0.9179496956941224
theta t = 0.37940866141050833
gammam : (1.7565123037446524+1213.6747027928786j)
Coefficient Gamma perpendiculaire = (-0.532674827074587+0.0005183080305073682j)
Coefficient de réflexion : (-0.4827445290353380 4+0.024638996445289867j)

d = 107.00467279516349
Coefficient de réflexion total = (-0.10088303669518639+0.023820492133112513j)

En = (0.0026374981020233063-0.0016172177586574461j) N/C
P = 2.047349701180745e-13 W
P = -96.8880797056569 dBm
```

FIGURE 29 – Résultats du code - Réflexion triple

B Codes

main.py

```
import matplotlib.pyplot as plt
from buildfield import buildfield

from position import Position
from emetteur import Emetteur
from heatmap_dBm import heatmap_dBm
from heatmap_Mbps import heatmap_Mbps

emetteurs = [Emetteur(Position(100,65))]
xrange = 200; yrange = 110;

building = buildfield(); plt.gca().invert_yaxis(); building.plot();

heatmap_dBm(emetteurs, xrange, yrange, building);
```

buildfield.py

```
import math
import cmath
from wall import Wall
from building import Building
from position import Position


def buildfield():
    positionsh = [[(85,0),(115,0)],[(10,10),(35,10)],[(165,10),(190,10)],
                  [(85,15),(90,15)],[(110,15),(115,15)],[(35,20),(90,20)],
                  [(110,20),(165,20)],[(0,45),(10,45)],[(190,45),(200,45)],
                  [(0,110),(35,110)],[(35,100),(95,100)],[(105,100),(165,100)
                  [(165,110),(200,110)]]
    positionsv = [[(0,45),(0,110)],[(10,10),(10,45)],[(35,10),(35,20)],
                  [(85,0),(85,15)],[(90,15),(90,20)],[(115,0),(115,15)],[(110
                  [(165,10),(165,20)],[(165,100),(165,110)],[(190,10),(190,45
                  [(200,45),(200,110)]]
    positionsch = [[(90,30),(110,30)],[(10,35),(13,35)],[(15,35),(35,35)],
                  [(110,40),(140,40)],[(10,45),(13,45)],[(15,45),(35,45)],
                  [(110,45),(140,45)],[(165,45),(185,45)],[(187,45),(190,45)]
                  [(140,50),(150,50)],[(152,50),(160,50)],[(35,60),(85,60)],
                  [(90,60),(95,60)],[(105,60),(140,60)],[(140,70),(160,70)],
                  [(0,80),(5,80)],[(10,80),(35,80)],[(37,80),(68,80)],[(70,80
                  [(105,80),(120,80)],[(125,80),(175,80)]]
    positionscv = [[(30,80),(30,95)],[(35,20),(35,35)],[(35,45),(35,80)],
                  [(75,80),(75,85)],[(75,95),(75,100)],[(85,60),(85,70)],
                  [(90,30),(90,40)],[(90,45),(90,60)],[(95,60),(95,80)],
                  [(105,60),(105,80)],[(110,30),(110,40)],[(110,45),(110,52)]
                  [(110,54),(110,60)],[(125,80),(125,85)],[(125,95),(125,100)
                  [(140,20),(140,40)],[(140,45),(140,50)],[(140,60),(140,70)]
                  [(165,20),(165,30)],[(165,40),(165,45)],[(175,45),(175,80)]

    walls = [];
    for pos in positionsh:
        walls.append(Wall(Position(pos[0][0],pos[0][1]),Position(pos[1][0],po
    for posv in positionsv:
        walls.append(Wall(Position(posv[0][0],posv[0][1]),Position(posv[1][0]
    for pos in positionsch:
        walls.append(Wall(Position(pos[0][0],pos[0][1]),Position(pos[1][0],po
```

```
for posv in positionscv:
    walls.append(Wall(Position(posv[0][0],posv[0][1]),Position(posv[1][0]
build = Building(walls);

return build;
```

building.py

```
import math
from wall import Wall;

class Building:
    def __init__(self,wallslist):
        self.wallslist = wallslist;

    def plot(self):
        for wall in self.wallslist:
            if wall.getmat() == "b":
                wall.plot();
            else:
                wall.plotred()

    def getwallslist(self):
        return self.wallslist

    def toString(self):
        mot = ""
        for wall in self.wallslist:
            mot += 'Mur␣:' + wall.toString()
        return mot
```

coefficients.py

```
import math
import cmath
from fonctions import distance

def transmission(wallslist, emetteur, recepteur):

    """ calcule et renvoie le produit des coefficients de transmission d'un r
        et le nombre de murs travers s par le rayon
    """

    d = 0
    coeff = 1
    for wall in wallslist:
        if wall.isintersect(emetteur.getposition(), recepteur.getposition()):

            theta_i = thetai(emetteur.getposition(), wall.intersection(emette
            theta_t = thetat(wall, theta_i)
            s = wall.getwidth()/math.cos(theta_t)

            coeff *= coefftransmission(wall, emetteur, recepteur, s)
            d += s

    return coeff, d


def coefftransmission(wall, emetteur, recepteur, s):

    """calcule et renvoie le coefficient de transmission d'un rayon traversan

    emetteur = emetteur.getposition()
    recepteur = recepteur.getposition()
    theta_i = thetai(emetteur, recepteur, wall)

    num = (1 - gammaperp(wall, theta_i)**2)*cmath.exp(-wall.gammam()*s)
    denom = 1 - ((gammaperp(wall, theta_i)**2)*cmath.exp(-2*wall.gammam()*s)*
    coefftransmission = num/denom
    return coefftransmission

def coeffreflexion(wall, emet, reflex):
```

```
    """calcule et renvoie le coefficient de r flexion d'un rayon sur un mur

    emetteur = emet.getposition()
    theta_i = thetai(emetteur.getposition(), reflex, wall)
    theta_t = thetat(wall, theta_i)
    gammap = gammaperp(wall, theta_i)
    s = abs(wall.getwidth()/math.cos(theta_t))

    numerator = gammap*cmath.exp(-2*wall.gammam()*s)*cmath.exp(complex(0, 2*w
    denominator = 1 - ((gammap**2)*cmath.exp(-2*wall.gammam()*s)*cmath.exp(co
    coeffreflexion = gammap + (1 - gammap**2)*(numerator/denominator)

    return coeffreflexion

def gammaperp(wall, theta_i):

    """calcule et renvoie le coefficient de r flexion pour la polarisation p

    Z1 = math.sqrt(wall.mu0/(wall.eps0))
    Z2 = wall.impedance()
    theta_t = thetat(wall, theta_i)
    gammaperp = (Z2*math.cos(theta_i) - Z1*math.cos(theta_t))/(Z2*math.cos(th
    return gammaperp

def thetai(posemetteur, posincidence, wall):

    """calcule et renvoie l'angle d'incidence avec un  metteur  et point d'in

    if wall.is_vertical():
        if posemetteur.getx() == posincidence.getx():
            theta_i = math.pi/2
        else:
            theta_i = abs(math.atan((posemetteur.gety() - posincidence.gety()
    else:
        if posemetteur.gety() == posincidence.gety():
            theta_i = math.pi/2
        else:
            theta_i = abs(math.atan((posemetteur.getx() - posincidence.getx()

    return theta_i
```

```
def thetat(wall, theta_i):

    """calcule et renvoie l'angle theta_t transmis via la loi de Snell-Descar

    theta_t = math.asin(math.sqrt(wall.eps0/wall.eps())*math.sin(theta_i))
    return theta_t
```

emetteur.py

```
from position import Position

class Emetteur:
    def __init__(self,position):
        self.position = position

    def getposition(self):
        return self.position
```

fonctions.py

```
from position import Position
import math

def distance(pos1,pos2):

    """calcule et renvoie la distance entre deux points"""

    d = math.sqrt((pos1.getx() - pos2.getx())**2 + (pos1.gety() - pos2.gety()
    return d

def pointimage(emetteur, wall):

    """calcule et renvoie le point ' metteur  image' d'un  metteur  en fonct

    if wall.is_vertical():
        y = emetteur.getposition().gety()
        d = emetteur.getposition().dist_w(wall)
        x = emetteur.getposition().getx() + 2*d
    else:
        x = emetteur.getposition().getx()
        d = emetteur.getposition().dist_w(wall)
        y = emetteur.getposition().gety() + 2*d
    mirror_position = Position(x,y)
    return mirror_position

def notonawall(building, point):

    """renvoie True si le point consid r  est sur/dans aucun des murs, fals

    bool = True
    for wall in building.wallslist:
        if wall.is_vertical():
            if point.getx() == wall.getstart().getx() and ((point.gety() > wa
                bool = False
                break
        else:
            if point.gety() == wall.getstart().gety() and ((point.getx() > wa
                bool = False
                break
```

```
return bool
```

heatmapdBm.py

```
import matplotlib.pyplot as plt
import math
from totalW import totalW
from totaldBm import totaldBm
from position import Position
from recepteur import Recepteur

def heatmap_dBm(emetteurs, x, y, building):

    """ calcule, affiche et renvoie sous forme de liste (de listes) la puissa
        de la carte pour une liste d'metteurs donn e.
    """

    final = [];
    nombre_emetteurs = len(emetteurs);

    if nombre_emetteurs == 1:
        for y in range(110):
            row = [];
            for x in range(200):
                recepteur = Recepteur(Position(x+0.5,y+0.5)) #on se positionn
                print(recepteur.getposition().toString())
                dBm = totaldBm(building, emetteurs[0], recepteur)
                row.append(dBm)
            final.append(row)

    else:
        for i in range(110):
            rowlist = 200 * [0]
            final.append(rowlist)
            n = 1;
        for emetteur in emetteurs:
            if n == nombre_emetteurs:
                for y in range(110):
                    for x in range(200):
                        recepteur = Recepteur(Position(x+0.5,y+0.5)) #on se p
                        print(str(n) + "␣" + str(recepteur.getposition().toSt
                        P = totalW(building, emetteur, recepteur)
```

```
                final[y][x] += P
                final[y][x] = 10*math.log(final[y][x]/10**(-3),10)
        else:
            for y in range(110):
                for x in range(200):
                    recepteur = Recepteur(Position(x+0.5,y+0.5)) #on se p
                    print(str(n) + "␣" + str(recepteur.getposition().toSt
                    P = totalW(building, emetteur, recepteur)
                    final[y][x] += P
        n += 1


hm=plt.imshow(final);
plt.colorbar(hm);
plt.clim(-82,-73)

return final
```

heatmapMbps.py

```
import matplotlib.pyplot as plt
from matplotlib import colors

def heatmap_Mbps(lis):

    """ calcule et renvoie sous forme de liste (de listes) le d bit binaire
        partir des dBm re us en chaque point de la carte.
    """

    final = [];
    for i in lis:
        row = [];
        for j in i:
            Mbps = (280/9)*j + (23320/9);
            row.append(Mbps)
        final.append(row)


    cmap = colors.ListedColormap(['black', 'red', 'orange', 'yellow', 'white'
    bounds = [0,10,80,160,240,320];
    norm = colors.BoundaryNorm(bounds, cmap.N)
    hm = plt.imshow(final, cmap=cmap, norm = norm);
    plt.colorbar(hm, cmap=cmap, norm=norm, boundaries=bounds);

    return final
```

position.py

```
import math
import matplotlib.pyplot as plt

class Position:
    def __init__(self,x,y):
        self.x = x;
        self.y = y;

    def dist_w(self,wall):
        if wall.is_vertical():
            res = (wall.getend().getx() - self.x)
        else :
            res = (wall.getend().gety() - self.y)
        return res

    def getx(self):
        return self.x;
    def gety(self):
        return self.y;
    def getposition(self):
        return self;

    def toString(self):
        return "(" + str(self.x) + "␣,␣"+str(self.y) + ")"



    def plot(self):
        posx = []
        posy = []
        posx.append(self.getx())
        posy.append(self.gety())
        plt.scatter(posx, posy)
        plt.show()

        """def distance_w(self,wall):
        if wall.is_vertical():
            res = abs(self.x-wall.getend().getx())
        else :
```

```
        res = abs(self.y-wall.getend().gety())
    return res"""
```

recepteur.py

```
from position import Position

class Recepteur:
    def __init__(self, position):
        self.position = position

    def getposition(self):
        return self.position
```

reflexion0.py

```
import matplotlib.pyplot as plt
import math
import cmath
from fonctions import distance
from coefficients import transmission

def reflexion0(wallslist, emetteur, recepteur):

    """ calcule et renvoie pour un  metteur  et un r cepteur donn s, la pui
        en prenant en compte les  vetuelles  transmissions.
        calcule aussi le champ  lectrique  en ce r cepteur.
    """

    he = 0.00353432
    beta = (1.696460031*(10**(11)))/299792458
    transmission1, n = transmission(wallslist, emetteur, recepteur)
    d = distance(recepteur.getposition(), emetteur.getposition()) + n

    if d == 0:
        d = 10**(-5)

    En = (transmission1*math.sqrt(60*1.7*0.1)*cmath.exp(complex(0,-d*beta)))/
    P = (he*abs(En))**2

    return P



def plot(emetteur, recepteur):
    startb = (emetteur.getposition().getx(),recepteur.getposition().getx());
    endb = (emetteur.getposition().gety(),recepteur.getposition().gety());
    plt.plot(startb,endb, color = 'yellow')
```

reflexion1.py

```
import matplotlib.pyplot as plt
import math
import cmath
from fonctions import distance, pointimage

from coefficients import transmission,coeffreflexion

def reflexion1(wallslist, emetteur, recepteur):

    """ calcule et renvoie pour un  metteur  et un r cepteur donn s, la pui
        (= somme) de toutes les ondes ayant subi une et une seule r flexion
        calcule aussi le champ  lectrique  total et le champ  lectrique  g r
    """

    P = 0
    Entot = 0

    for wall in wallslist:
        mirror = pointimage(emetteur, wall)
        if wall.isintersect(mirror.getposition(), recepteur.getposition()):
            reflex = wall.intersection(mirror.getposition(),recepteur.getposi

            transmission1, m = transmission(wallslist, emetteur, reflex)
            transmission2, n = transmission(wallslist, reflex, recepteur)
            coeffreflexion1 = coeffreflexion(wall, emetteur, reflex)
            coefftot = coeffreflexion1*transmission1*transmission2

            d = distance(recepteur.getposition(), mirror) + (m+n)

            En = (coefftot*math.sqrt(60*1.7*0.1)*cmath.exp(complex(0,-d*wall.
            Entot += En
            P += (wall.he*abs(En))**2

    return P


def plot(emetteur, recepteur, reflex):
    startb = (emetteur.getposition().getx(),reflex.getx());
    endb = (emetteur.getposition().gety(),reflex.gety());
```

```
plt.plot(startb,endb, color = 'green')

startb = (reflex.getx(),recepteur.getposition().getx());
endb = (reflex.gety(),recepteur.getposition().gety());
plt.plot(startb,endb, color = 'green')
```

reflexion2.py

```
import matplotlib.pyplot as plt
import math
import cmath
from fonctions import distance, pointimage
from coefficients import transmission, coeffreflexion

def reflexion2(wallslist, emetteur, recepteur):

    """ calcule et renvoie pour un  metteur  et un r cepteur donn s, la pui
        (= somme) de toutes les ondes ayant subi deux r flexions en prenant
        calcule aussi le champ  lectrique  total et le champ  lectrique  g n
    """

    P = 0
    Entot = 0

    for wall1 in wallslist:
        mirror1 = pointimage(emetteur, wall1)
        for wall2 in wallslist:
            mirror2 = pointimage(mirror1, wall2)
            if wall2.isintersect(mirror2, recepteur.getposition()):
                reflex2 = wall2.intersection(mirror2, recepteur.getposition()
                if wall1.isintersect(mirror1, reflex2):
                    reflex1 = wall1.intersection(mirror1, reflex2)

                    transmission1, l = transmission(wallslist, emetteur, refl
                    transmission2, m = transmission(wallslist, reflex1, refle
                    transmission3, n = transmission(wallslist, reflex2, recep
                    coeffreflexion1 = coeffreflexion(wall1, emetteur, reflex1
                    coeffreflexion2 = coeffreflexion(wall2, reflex1, reflex2)
                    coefftot = coeffreflexion1*coeffreflexion2*transmission1*

                    d = distance(recepteur.getposition(),mirror2) + (l+m+n)

                    En = (coefftot*math.sqrt(60*1.7*0.1)*cmath.exp(complex(0,
                    Entot += En
                    P += (wall1.he*abs(En))**2
    return P
```

```
def plot(emetteur, recepteur, reflex1, reflex2):
    startb = (emetteur.getposition().getx(),reflex1.getx());
    endb =(emetteur.getposition().gety(),reflex1.gety());
    plt.plot(startb,endb, color = 'red')

    startb = (reflex1.getx(),reflex2.getx());
    endb =(reflex1.gety(),reflex2.gety());
    plt.plot(startb,endb, color = 'red')

    startb = (reflex2.getx(),recepteur.getposition().getx());
    endb =(reflex2.gety(),recepteur.getposition().gety());
    plt.plot(startb,endb, color = 'red')
```

reflexion3.py

```
import matplotlib.pyplot as plt
import math
import cmath
from fonctions import distance, pointimage
from coefficients import transmission, coeffreflexion

def reflexion3(wallslist, emetteur, recepteur):

    """ calcule et renvoie pour un  metteur  et un r cepteur donn s, la pui
        (= somme) de toutes les ondes ayant subi trois r flexions en prenant
        calcule aussi le champ  lectrique  total et le champ  lectrique  g n
    """

    P = 0
    Entot = 0

    for wall1 in wallslist:
        mirror1 = pointimage(emetteur, wall1)
        for wall2 in wallslist:
            mirror2 = pointimage(mirror1, wall2)
            for wall3 in wallslist:
                mirror3 = pointimage(mirror2, wall3)
                if wall3.isintersect(recepteur.getposition(), mirror3):
                    reflex3 = wall3.intersection(recepteur.getposition(), mir
                    if wall2.isintersect(reflex3, mirror2):
                        reflex2 = wall2.intersection(mirror2.getposition(),re
                        if wall1.isintersect(reflex2, mirror1):
                            reflex1 = wall1.intersection(mirror1.getposition(

                            transmission1, k = transmission(wallslist, emette
                            transmission2, l = transmission(wallslist, reflex1
                            transmission3, m = transmission(wallslist, reflex2
                            transmission4, n = transmission(wallslist, reflex3
                            coeffreflexion1 = coeffreflexion(wall1, emetteur,
                            coeffreflexion2 = coeffreflexion(wall2, reflex1,
                            coeffreflexion3 = coeffreflexion(wall3, reflex2,
                            coefftot = coeffreflexion1*coeffreflexion2*coeffr
```

```
                                d = distance(recepteur.getposition(), mirror3) +

                                En = (coefftot*math.sqrt(60*1.7*0.1)*cmath.exp(co
                                Entot += En
                                P += (wall1.he*abs(En))**2
    return P


def plot(emetteur, recepteur, reflex1, reflex2, reflex3):
    startb = (emetteur.getposition().getx(),reflex1.getx());
    endb =(emetteur.getposition().gety(),reflex1.gety());
    plt.plot(startb,endb, color = 'blue')

    startb = (reflex1.getx(),reflex2.getx());
    endb =(reflex1.gety(),reflex2.gety());
    plt.plot(startb,endb, color = 'blue')

    startb = (reflex2.getx(),reflex3.getx());
    endb =(reflex2.gety(),reflex3.gety());
    plt.plot(startb,endb, color = 'blue')

    startb = (reflex3.getx(),recepteur.getposition().getx());
    endb =(reflex3.gety(),recepteur.getposition().gety());
    plt.plot(startb,endb, color = 'blue')
```

totaldBm.py

```
import math
from reflexion0 import reflexion0
from reflexion1 import reflexion1
from reflexion2 import reflexion2
from reflexion3 import reflexion3
from fonctions import notonawall

def totaldBm(building, emetteur, recepteur):

    """calcule la puissance totale re ue en dBm en un r cepteur donn ."""

    wallslist = building.wallslist
    dBm = 0;
    reflex0 = 0;
    reflex1 = 0;
    reflex2 = 0;
    reflex3 = 0;
    if notonawall(building, recepteur.getposition()) and notonawall(building,
        reflex0 = reflexion0(wallslist, emetteur, recepteur)
        reflex1 = reflexion1(wallslist, emetteur, recepteur)
        reflex2 = reflexion2(wallslist, emetteur, recepteur)
        reflex3 = reflexion3(wallslist, emetteur, recepteur)

    Ptot = (1/(8*73))*(reflex0+reflex1+reflex2+reflex3)

    if Ptot > 10**(-250):
        dBm = 10*math.log(Ptot/10**(-3),10)
    return dBm
```

totalW.py

```
import math
from reflexion0 import reflexion0
from reflexion1 import reflexion1
from reflexion2 import reflexion2
from reflexion3 import reflexion3
from fonctions import notonawall

def totalW(building, emetteur, recepteur):

    """calcule la puissance totale reçue en W en un récepteur donné."""

    wallslist = building.wallslist
    dBm = 0;
    reflex0 = 0;
    reflex1 = 0;
    reflex2 = 0;
    reflex3 = 0;

    if notonawall(building, recepteur.getposition()) and notonawall(building,
        reflex0 = reflexion0(wallslist, emetteur, recepteur)
        reflex1 = reflexion1(wallslist, emetteur, recepteur)
        reflex2 = reflexion2(wallslist, emetteur, recepteur)
        reflex3 = reflexion3(wallslist, emetteur, recepteur)

    Ptot = (1/(8*73))*(reflex0+reflex1+reflex2+reflex3)
    return Ptot
```

wall.py

```
import math
import cmath
import matplotlib.pyplot as plt
from position import Position

class Wall:
    def __init__(self, start, end, width, mat):
        self.start = start
        self.end = end
        self.width = width
        self.mat = mat                                  #b = b ton; br = bri

    he = 0.00353432
    mu0 = 4*math.pi*(10**(-7))
    eps0 = 8.85418782*(10**(-12))
    w = 1.696460031*(10**(11))                          #2pif
    beta = (1.696460031*(10**(11)))/299792458           #w/c

    def is_vertical(self):

        """renvoie True si un mur est vertical, False sinon"""

        res = True;
        if self.start.gety() == self.end.gety():
            res = False;
        return res

    def intersection(self, pos1, pos2):

        """calcule et renvoie l'intersection entre un mur et un segment de dr

        x0 = pos1.getx(); y0 = pos1.gety();
        x1 = pos2.getx(); y1 = pos2.gety();

        if self.is_vertical():
            x = self.getstart().getx()
            y = ((y1-y0)/(x1-x0))*(x - x0) + y0
        else:
```

```
            y = self.getstart().gety()
            x = (y-y0)*((x1-x0)/(y1-y0)) + x0
        inter = Position(x,y)

        return inter

    def isintersect (self, pos1, pos2):

        """renvoie True si le segment de droite entre deux points croise un n

        bool = False;
        if self.is_vertical():
            if pos1.getx() != pos2.getx():
                inter = self.intersection(pos1,pos2)
                if (pos1.getx() > inter.getx() and pos2.getx() < inter.getx()
                    if inter.gety() > self.getstart().gety() and inter.gety()
                        bool = True
        else:
            if pos1.gety() != pos2.gety():
                inter = self.intersection(pos1,pos2)
                if (pos1.gety() > inter.gety() and pos2.gety() < inter.gety()
                    if inter.getx() > self.getstart().getx() and inter.getx()
                        bool = True
        return bool

    def eps(self):
        if self.mat == "b":
            eps = self.eps0*5
        if self.mat == "br":
            eps = self.eps0*4.6
        if self.mat == "c":
            eps = self.eps0*2.25
        return eps

    def sigma(self):
        if self.mat == "b":
            sigma = 0.014
        if self.mat == "br":
            sigma = 0.02
        if self.mat == "c":
            sigma = 0.04
        return sigma
```

```
def impedance(self):
    Z = cmath.sqrt(self.mu0/(complex(self.eps(), - (self.sigma()/self.w))
    return Z

def gammam(self):
    gammam = complex(0, self.w*cmath.sqrt(self.mu0*(complex(self.eps(), -
    return gammam

def getstart(self):
    return self.start
def getend(self):
    return self.end
def getwidth(self):
    return self.width
def getmat(self):
    return self.mat

def toString(self):
    details = "Start␣"+self.start.toString()+ "␣End␣" + self.end.toString
    return details
def plot(self):
    startb = (self.start.getx(),self.end.getx());
    endb =(self.start.gety(),self.end.gety());
    plt.plot(startb,endb, color = 'black')
    return
def plotred(self):
    startb = (self.start.getx(),self.end.getx());
    endb =(self.start.gety(),self.end.gety());
    plt.plot(startb,endb, color = 'red')
    return
```