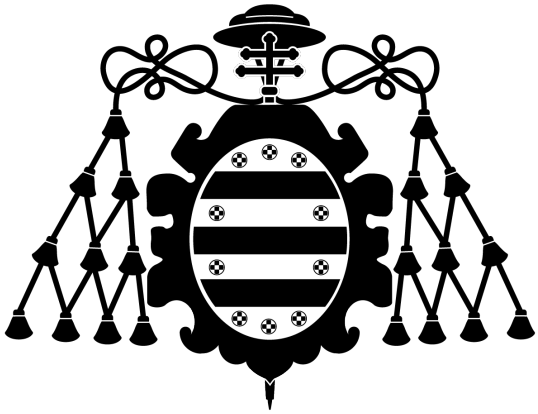

Universidad de Oviedo

Departamento de Ingeniería Eléctrica, Electrónica, de Computadores y Sistemas

Área de Teoría de la Señal y Comunicaciones

Herramientas eficientes de estimación de propagación mediante técnicas de fuentes equivalentes, ANN y computación en GPU

ANEXOS

Autor: David Martínez Álvarez

Director: Fernando Las-Heras Andrés

ÍNDICE DE CONTENIDOS - ANEXOS

ÍNDICE DE FIGURAS - ANEXOS

ÍNDICE DE TABLAS - ANEXOS

Anexo 1 Fundamentos teóricos

A continuación se desglosan los diferentes conceptos teóricos básicos que sirven de base para el desarrollo realizado en el presente trabajo.

Anexo 1.1 Introducción a la radiación por densidades de corriente

Como es bien sabido, la teoría de la propagación electromagnética tiene su base en las cuatro ecuaciones de Maxwell, que descritas en su forma diferencial tienen la forma indicada en las ecuaciones (***Ecuación 1-1***) a (*Ecuación 1-4*).

$$\nabla \times E = -M_i - \frac{\partial B}{\partial t} \qquad \textit{Ecuación 1-1}$$

$$\nabla \times H = J_i + J_c + \frac{\partial D}{\partial t} \qquad \textit{Ecuación 1-2}$$

$$\nabla \cdot D = q_{ev} \qquad \textit{Ecuación 1-3}$$

$$\nabla \cdot B = q_{mv} \qquad \textit{Ecuación 1-4}$$

Estas ecuaciones, además de para tener en cuenta los campos electromagnéticos producidos por las corrientes eléctricas en movimiento, sirven de base para definir una variable ficticia[1], denominada densidad de corriente magnética **M**, que representa la forma en la que las cargas magnéticas, de nuevo ficticias, fluirían a lo largo de un material en el que existan campos electromagnéticos [21].

Tal y como explica C. A. Balanis en el capítulo 6 de su libro [21], realizando operaciones matemáticas de diversa índole sobre las ecuaciones (***Ecuación 1-1***) a (*Ecuación 1-4*), puede llegar a definirse la radiación de campos electromagnéticos, producida por una densidad de corriente magnética **M** arbitraria, quedando de la forma indicada en las ecuaciones (***Ecuación 1-5***) a (*Ecuación 1-7*).

[1] Se adjetiva como ficticia porque esta densidad de corriente magnética no puede encontrarse físicamente, pues es sabido que no existen las cargas magnéticas como tales, o al menos, aún no ha sido probada su existencia. Este término, es un formalismo matemático utilizado para balancear adecuadamente las ecuaciones de Maxwell.

$$E_{M,x} = \frac{-1}{4\pi}\int_{s'}\begin{Bmatrix} M_y[z-z'] \\ M_z[-(y-y')] \end{Bmatrix}\left(\frac{1+j\beta R}{R^3}\right)e^{-j\beta R}dS' \qquad \textit{Ecuación 1-5}$$

$$E_{M,y} = \frac{-1}{4\pi}\int_{s'}\begin{Bmatrix} M_x[-(z-z')] \\ M_z[x-x'] \end{Bmatrix}\left(\frac{1+j\beta R}{R^3}\right)e^{-j\beta R}dS' \qquad \textit{Ecuación 1-6}$$

$$E_{M,z} = \frac{-1}{4\pi}\int_{s'}\begin{Bmatrix} M_x[y-y'] \\ M_y[-(x-x')] \end{Bmatrix}\left(\frac{1+j\beta R}{R^3}\right)e^{-j\beta R}dS' \qquad \textit{Ecuación 1-7}$$

Donde:

$$\overline{M} = M_X \cdot \hat{x} + M_Y \cdot \hat{y} + M_Z \cdot \hat{z}$$

$\bar{r} = x \cdot \hat{x} + y \cdot \hat{y} + z \cdot \hat{z}$ Posición del observador, donde se calcula el campo

$\bar{r}' = x' \cdot \hat{x} + y' \cdot \hat{y} + z' \cdot \hat{z}$ Posición de la fuente, que ocupa un volumen V', que en este caso degenera en una superficie S' (todas las corrientes se encuentran sobre dicha superficie)

$$R = |\bar{r} - \bar{r}'|$$

Estas ecuaciones servirán de base matemática para representar la radiación producida por un sistema arbitrario, de una forma más sencilla y fácil de manejar, lo que reducirá enormemente la complejidad matemática de dicho sistema.

Anexo 1.2 Efectos de la propagación en entornos con obstáculos

Cuando en un sistema en el que se trabaja con una fuente arbitraria de campos electromagnéticos, definida ésta por sus corrientes (eléctricas o magnéticas) de excitación, se tienen condiciones de espacio libre[2], se pueden definir los niveles de campo eléctrico o magnético en un punto arbitrario sin más que recurrir a la resolución de las ecuaciones de Maxwell (***Ecuación 1-1***) a (*Ecuación 1-4*), a las ecuaciones de radiación de densidades de corriente magnética (***Ecuación 1-5***) a (*Ecuación 1-7*), o a sus homónimas para el caso de densidades de corriente eléctrica. Sin embargo en el ámbito de una propagación en un entorno con obstáculos, donde el entorno de la fuente de campo está lleno de objetos que perturban la propagación, deben tenerse en cuenta para calcular los niveles de campo eléctrico o magnético en un punto las contribuciones que se producen por la interacción de las ondas electromagnéticas en los mencionados obstáculos. Estas interacciones con los obstáculos son principalmente las reflexiones y las difracciones.

Existen numerosos métodos que estudian el comportamiento de las ondas electromagnéticas frente a estos efectos de la propagación en medios con obstáculos, como pueden ser las técnicas de Óptica Geométrica [23], [24], [25], o las técnicas de Óptica Física [26], que serán las utilizadas en la presente investigación, o también el Método de los Momentos [27], [28], que si bien es un método exacto en cuanto a que no realiza más aproximaciones en su resolución que las necesarias para su evaluación numérica, su coste computacional para estructuras eléctricamente grandes lo vuelve inutilizable para situaciones medianamente complejas.

Asimismo, cabe comentar que existen numerosos desarrollos de software que tratan el problema de la propagación en entornos de muchos obstáculos como puede ser por ejemplo un escenario de interior o INDOOR. El objetivo último de la presente investigación será el de desarrollar un software que proporcione unos resultados lo más precisos posible, a la vez que reduzca la carga computacional asociada al problema.

Anexo 1.2.1 Reflexión y Transmisión de frentes de onda planos.

El fenómeno de la reflexión/transmisión, no es más que el efecto que se produce cuando una onda electromagnética que viaja por un medio choca con una superficie arbitraria que presenta frontera con el medio por el que viajaba dicha onda, y a raíz de este choque se

[2] Se consideran condiciones de espacio libre, cuando no existen obstáculos que alteren el comportamiento de la propagación de las ondas electromagnéticas con respecto a las condiciones de propagación en el vacío, o en el caso de que existan, su contribución es despreciable.

producen dos nuevas ondas electromagnéticas, una reflejada, y otra transmitida. La onda reflejada se mantendrá en el medio original por el que fluía la onda original, y la onda transmitida pasará al segundo medio cuya superficie frontera con el primer medio es la causante de todos estos efectos. Estas ondas reflejada y transmitida, se rigen por las leyes conocidas como leyes de Snell.

$$\theta_r = \theta_i \qquad \text{(Ley de Snell de la Reflexión)} \qquad \textit{Ecuación 1-8}$$

$$\beta_1 \sin\theta_i = \beta_2 \sin\theta_t \qquad \text{(Ley de Snell de la Refracción)} \qquad \textit{Ecuación 1-9}$$

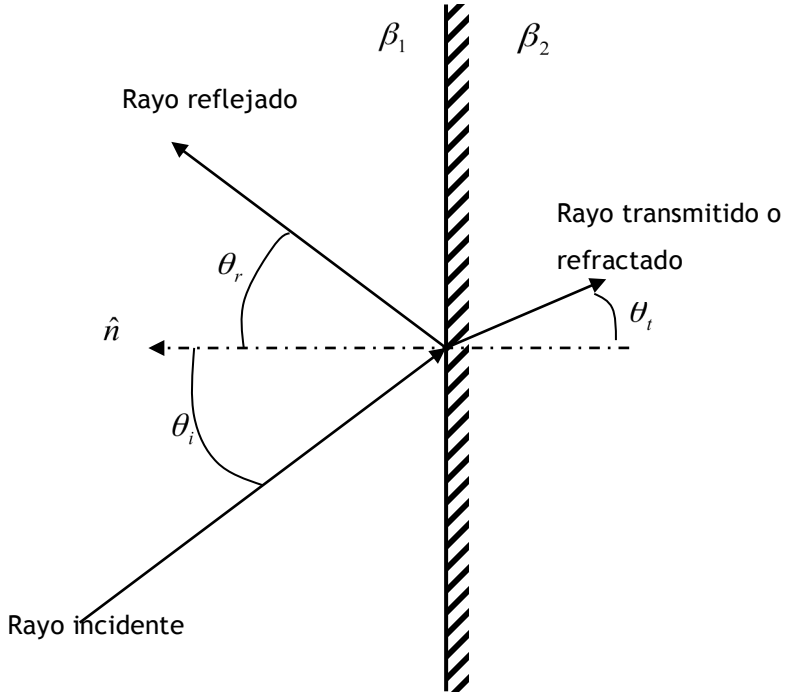

Figura 1: Leyes de Snell

Anexo 1.2.2 Difracción

La difracción es el fenómeno por el que, cuando una onda electromagnética incide sobre un obstáculo, este se convierte en una nueva fuente de ondas electromagnéticas. Este fenómeno se produce por la generalización del principio de Fermat, que dice que "*el camino entre dos puntos dados que recorre un rayo de luz es tal que para ese camino, el tiempo que tarda la luz en recorrerlo es mínimo*". Esta generalización se basa en la discontinuidad producida en el camino del rayo de luz entre dos puntos, debida a la presencia de un obstáculo. Cuando las condiciones del obstáculo sobre el que incide la onda cumplen ciertos parámetros, este efecto produce variaciones de campo lo suficientemente grandes como para modificar en gran medida el comportamiento de la onda electromagnética. Particularmente importante, es la difracción denominada en “filo de cuchillo” o en “esquina”, que tiene lugar cuando la incidencia de la onda electromagnética tiene lugar sobre una superficie en forma de arista[3] cuyas dimensiones son superiores a varias longitudes de onda.

[3] Una arista, como por ejemplo las esquinas de un edificio.

Anexo 1.3 Introducción a la Óptica Geométrica

Existen diversos métodos matemáticos que estudian los fenómenos de reflexión y difracción. Especialmente útil por su versatilidad y precisión resulta la teoría clásica de óptica geométrica para alta frecuencia, y sus extensiones para añadir los fenómenos de difracción, como son la "Teoría Geométrica de la Difracción" (GTD - Geometric Theory of Diffraction) [24], y su extensión, la "Teoría Uniforme de la Difracción" (UTD - Uniform Theory of Diffraction) [25].

Originalmente, la Óptica Geométrica fue desarrollada para analizar la propagación de la luz, es decir, a frecuencias suficientemente altas como para poder considerar la propagación de la onda como si fuera un rayo que se propaga por el camino más corto entre dos puntos cualesquiera[4]. Estos rayos están sujetos a las ya comentadas leyes de Snell[5], que definen la interacción del rayo de luz con los obstáculos que se encuentra en su camino. Partiendo de las ecuaciones de Maxwell, y bajo las mismas consideraciones realizadas para la luz, puede llegarse a las ecuaciones para el campo eléctrico producido por la reflexión y la difracción que tienen lugar sobre respectivamente, una pared plana, y una esquina en "filo de cuchillo" [21].

Reflexión:

$$E^r(s) = E^i{}_0 \cdot \overline{R} \cdot \sqrt{\frac{\rho_1 \rho_2}{(\rho_1 + s)(\rho_2 + s)}} \cdot e^{-j\beta s} \qquad \textit{Ecuación 1-10}$$

Donde:

$E^i{}_0$	Es el campo eléctrico incidente en el punto de reflexión.
ρ_1 y ρ_2	Son los factores de atenuación espacial, que dependen del tipo de fuente inicial con la que se trabaja.
$\overline{R}$	Es el coeficiente de reflexión de Fresnell.
s	*Es la distancia del punto de reflexión al punto de observación*

[4] Siguiendo el principio de Fermat.

[5] Apartado Anexo 1.2.1

Difracción:

$$E^r(s) = E^i{}_0 \cdot \overline{D} \cdot A(\rho_c, s) \cdot e^{-j\beta s}$$ *Ecuación 1-11*

Donde:

$E^i{}_0$	Es el campo eléctrico incidente en el punto de difracción.
$A(\rho_c, s)$	Es el factor de atenuación espacial, que depende del tipo de obstáculo sobre el que se incide, de la posición del punto sobre el que calcular el campo, y del tipo de fuente inicial.
$\overline{D}$	Es el coeficiente de Difracción del obstáculo sobre el que se produce el efecto

Anexo 1.4 Introducción a la Óptica Física

La Óptica Física es una técnica asintótica de alta frecuencia basada en una sencilla aproximación desarrollada sobre las corrientes inducidas en la superficie de un objeto sobre el que incide una onda electromagnética. La Óptica Física está basada, al igual que la Óptica Geométrica, en las ecuaciones de Maxwell pero en este caso expresadas de un modo ligeramente diferente a las consideradas en las ecuaciones (***Ecuación 1-1***) a (*Ecuación 1-4*). Suponiendo dichas ecuaciones formuladas para un medio lineal e isótropo y considerando régimen sinusoidal permanente, es decir, la variación temporal se produce conforme a $e^{j\omega t}$, las ecuaciones de Maxwell se formularán según se indica en (***Ecuación 1-12***) - (*Ecuación 1-15*).

$$\nabla \cdot \overline{E} = \frac{\rho}{\varepsilon} \qquad \textit{Ecuación 1-12}$$

$$\nabla \times \overline{E} = -j\omega\mu \cdot \overline{H} \qquad \textit{Ecuación 1-13}$$

$$\nabla \cdot \overline{H} = 0 \qquad \textit{Ecuación 1-14}$$

$$\nabla \times \overline{H} = \overline{J} + j\omega\varepsilon \cdot \overline{E} \qquad \textit{Ecuación 1-15}$$

Anexo 1.4.1 El concepto de potencial vector magnético

De la misma forma que cuando se introduce el concepto de corriente magnética, cuando se pretenden resolver las ecuaciones (***Ecuación 1-12***) a (*Ecuación 1-15*), resulta de utilidad introducir un mecanismo matemático que se basa en la definición de una variable auxiliar que no tiene equivalencia física real, pero que permite simplificar el problema de tal forma que gracias a ella la resolución de las ecuaciones resulta mucho más sencilla. En este caso, se introduce el concepto de potencial vector magnético ($\bar{A}$), que se define de la forma indicada en (***Ecuación 1-16***).

$$\overline{H} = \frac{1}{\mu} \cdot \nabla \times \overline{A} \qquad \textit{Ecuación 1-16}$$

$\bar{A}$ es empleado para resolver matemáticamente las ecuaciones de Maxwell cuando éstas están escritas con cargas y corrientes eléctricas, como es el caso del presente desarrollo.

El campo magnético $\overline{H}$, definido en (*Ecuación 1-16*), cumple directamente la ecuación (*Ecuación 1-14*), ya que matemáticamente:

$$\nabla \cdot \left(\nabla \times \overline{A} \right) = 0 \qquad \textit{Ecuación 1-17}$$

Pero la condición impuesta por la ecuación (*Ecuación 1-16*) no sólo es verificada por una solución cualquiera de $\bar{A}$, sino también por $\bar{A} \pm \nabla \xi$, con lo que se comprueba que existe un grado de libertad en la elección de $\bar{A}$. Esto es de mucha utilidad en una etapa más avanzada de este desarrollo, ya que este grado de libertad es necesario para la resolución de la ecuación (*Ecuación 1-26*).

Introduciendo (*Ecuación 1-16*) en (*Ecuación 1-13*), se llega a:

$$\nabla \times \overline{E} = -j\omega\mu \cdot \left[\frac{1}{\mu} \cdot \nabla \times \overline{A} \right] = -j\omega \cdot \left(\nabla \times \overline{A} \right) \qquad \textit{Ecuación 1-18}$$

de donde:

$$\nabla \times \left(\overline{E} + j\omega \overline{A} \right) = 0 \qquad \textit{Ecuación 1-19}$$

Dado que:

$$\nabla \times \left(\nabla \varphi \right) = 0 \qquad \textit{Ecuación 1-20}$$

se puede representar lo que está entre paréntesis en la ecuación (*Ecuación 1-19*) como el gradiente de una función ϕ, que se denomina potencial escalar eléctrico:

$$\nabla \phi = -j\omega \cdot \overline{A} - \overline{E} \qquad \textit{Ecuación 1-21}$$

El signo negativo se le añade por convenio.

Despejando de la ecuación anterior, (*Ecuación 1-21*), el campo eléctrico, se obtiene:

$$\overline{E} = -\nabla \phi - j\omega \overline{A} \qquad \textit{Ecuación 1-22}$$

e introduciendo este resultado en la ecuación (*Ecuación 1-15*):

$$\nabla \times \bar{H} = \bar{J} + j\omega\varepsilon \cdot \left(-\nabla\phi - j\omega\bar{A}\right) = \bar{J} - j\omega\varepsilon \cdot \nabla\phi + \omega^2 \varepsilon \cdot \bar{A} \qquad \text{Ecuación 1-23}$$

Por otra parte, de (*Ecuación 1-16*) se sigue que:

$$\nabla \times \bar{H} = \nabla \times \left(\frac{1}{\mu} \cdot \nabla \times \bar{A}\right) = \frac{1}{\mu} \cdot \left(\nabla\left(\nabla \cdot \bar{A}\right) - \nabla^2 \bar{A}\right) \qquad \text{Ecuación 1-24}$$

Entonces, de (*Ecuación 1-23*) y (*Ecuación 1-24*):

$$\nabla\left(\nabla \cdot \bar{A}\right) - \nabla^2 \bar{A} = \mu \cdot \bar{J} - j\omega\varepsilon\mu \cdot \nabla\phi + \omega^2 \varepsilon\mu \cdot \bar{A} \qquad \text{Ecuación 1-25}$$

$$\nabla^2 \bar{A} - \nabla\left(\nabla \cdot \bar{A} - j\omega\varepsilon\mu \cdot \nabla\phi\right) + \omega^2 \varepsilon\mu \cdot \bar{A} = -\mu \cdot \bar{J} \qquad \text{Ecuación 1-26}$$

Aplicando la condición de separabilidad de Lorenz, dado que, como ya se subrayó anteriormente, existe un grado de libertad en la elección de $\bar{A}$, se puede igualar a cero lo incluido dentro del paréntesis de la ecuación (*Ecuación 1-26*). Queda, pues:

$$\nabla^2 \bar{A} + \omega^2 \varepsilon\mu \cdot \bar{A} = -\mu \cdot \bar{J} \qquad \text{Ecuación 1-27}$$

cuya solución se expresa de la forma:

$$\bar{A} = \frac{\mu}{4\pi} \cdot \iiint_V \frac{\bar{J} \cdot e^{-jkr}}{r} dV \qquad \text{Ecuación 1-28}$$

A partir del valor de $\bar{A}$ así calculado, los campos eléctrico y magnético se expresan tal y como indica la ecuación (*Ecuación 1-29*):

$$\bar{E} = \frac{1}{j\omega\mu\varepsilon} \cdot \nabla \times \left(\nabla \times \bar{A}\right)$$

$$\bar{H} = \frac{1}{\mu} \cdot \nabla \times \bar{A} \qquad \text{Ecuación 1-29}$$

En la se representan los vectores que intervienen en el cálculo del potencial vector magnético:

- $\bar{r}'$ es el vector de posición de un punto genérico contenido dentro del volumen V.

- $\bar{J}(\bar{r}')$ es el vector "corriente eléctrica de volumen" correspondiente al punto determinado por el vector de posición $\bar{r}'$.
- $\bar{r}$ es el vector de posición de un punto de observación P. Su correspondiente vector unitario, $\hat{r}$, es el que determina la dirección de observación.

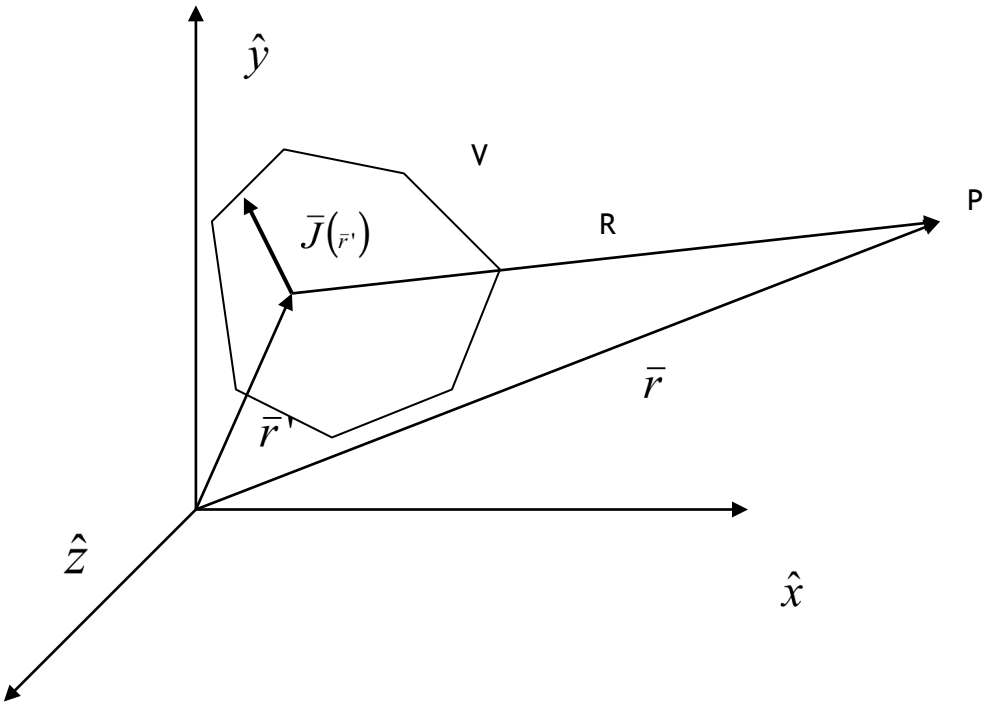

Figura 2: Vectores para el cálculo del potencial vector magnético

Si se pretende calcular el campo en el punto P, la ecuación (*Ecuación 1-28*) se transforma en:

$$\bar{A} = \frac{\mu}{4\pi} \cdot \iiint_V \frac{\bar{J}(\bar{r}') \cdot e^{-jkR}}{R} \cdot dV \qquad \textit{Ecuación 1-30}$$

siendo $R = |\bar{r} - \bar{r}'|$

Anexo 1.4.2 Aproximación de campo lejano

Se realizan cálculos en la situación de campo lejano cuando los puntos de observación están a una distancia lo suficientemente grande del elemento que crea dichos campos, de modo que permita realizar las dos suposiciones siguientes:

- El vector posición del punto de observación, $\bar{r}$, es mucho mayor que cualquiera de los vectores $\bar{r}'$ que determinan puntos del volumen cargado eléctricamente.
- La onda electromagnética que se propaga se puede considerar que es una onda plana, con frente de onda perpendicular a la dirección de propagación, que se considerará coincidente con la dirección de observación $\hat{r}$.

La primera de las dos condiciones, la relativa a la magnitud de los vectores de posición del punto de observación y del punto que crea el campo, lleva a dos actuaciones sobre la parte de la ecuación contenida dentro de la integral:

- En lo referente a la fase $e^{-jk|\bar{r}-\bar{r}'|}$:

$$|\bar{r}-\bar{r}'|^2 = |\bar{r}|^2 + |\bar{r}'|^2 - 2\cdot\bar{r}\cdot\bar{r}'$$

$$|\bar{r}-\bar{r}'| = r\cdot\sqrt{1+\frac{r'^2}{r^2}-\frac{2\cdot\hat{r}\cdot\bar{r}'}{r}} \cong$$

$$\cong r\cdot\sqrt{1+0-\frac{2\cdot\hat{r}\cdot\bar{r}'}{r}}$$

como $\lim_{x\to 0}\sqrt{1-x} \approx 1-\frac{x}{2}$

entonces

$$|\bar{r}-\bar{r}'| \cong r\cdot\left(1-\frac{2\cdot\hat{r}\cdot\bar{r}'}{r}\right) = r-\hat{r}\cdot\bar{r}' \qquad \textit{Ecuación 1-31}$$

- En lo relacionado con el módulo R:

$$R = |\bar{r}-\bar{r}'| \cong |\bar{r}| = r \qquad \textit{Ecuación 1-32}$$

La ecuación (*Ecuación 1-30*), aplicándole (*Ecuación 1-31*) y (*Ecuación 1-32*), queda de la forma:

$$\overline{A} = \frac{\mu}{4\pi} \cdot \frac{e^{-jkr}}{r} \cdot \iiint_V \overline{J}(\overline{r}') \cdot e^{-jk\overline{r}'\cdot\hat{r}} \cdot dV \qquad \textit{Ecuación 1-33}$$

Sustituyendo esta expresión en (*Ecuación 1-29*) y tras algunas manipulaciones realizadas teniendo en cuenta las aproximaciones empleadas para los términos en $|\overline{r} - \overline{r}'|$ se puede llegar a:

$$\overline{H} = -\frac{j\omega}{\eta} \cdot \left(\hat{r} \times \overline{A}\right) \qquad \textit{Ecuación 1-34}$$

$$\overline{E} = -j\omega \cdot \left(\hat{r} \times \overline{A}\right) \times \hat{r} \qquad \textit{Ecuación 1-35}$$

Las expresiones anteriores ponen de manifiesto que:

$$\overline{E} = \mu \cdot \left(\overline{H} \times \hat{r}\right) \qquad \textit{Ecuación 1-36}$$

lo cual verifica el comportamiento del campo lejano en el sentido de que $\overline{E}$ y $\overline{H}$, y la dirección de propagación, tienen la misma relación que las ondas planas.

En el caso que se trata en la presente investigación, se puede realizar una simplificación más en la formulación del vector potencial magnético. Esta no es más que recordar que las corrientes eléctricas que circulan en las superficies conductoras, son superficiales y no volumétricas. Teniendo esto en cuenta, la ecuación (*Ecuación 1-33*) toma la forma:

$$\overline{A} = \frac{\mu}{4\pi} \cdot \frac{e^{-jkr}}{r} \cdot \iint_S \overline{J}_S(\overline{r}') \cdot e^{-jk\overline{r}'\cdot\hat{r}} \cdot dS \qquad \textit{Ecuación 1-37}$$

donde S es la superficie en la que se realiza la integral, es decir, es la superficie de la pared sobre la que rebota la onda electromagnética.

Anexo 1.4.3 Aproximación de Óptica Física.

La Óptica Física se caracteriza por realizar la suposición de que cada punto de la superficie reflectora dispersa la onda incidente como si ésta incidiese en un plano tangente a la superficie en dicho punto. Esto se traduce en aproximar la corriente eléctrica en la superficie del reflector por:

$$\bar{J}_S = 2\cdot\left(\hat{n}_S \times \overline{H}_{inc}\right) \qquad \textit{Ecuación 1-38}$$

donde $\hat{n}_s$ es el vector normal a la superficie en el punto en el que el campo magnético incidente, $\overline{H}_{inc}$, alcanza a la pared.

Aplicando a la ecuación (*Ecuación 1-37*) la aproximación de campo lejano (*Ecuación 1-36*) y la de Óptica Física (*Ecuación 1-38*), se llega a las formulaciones de los campos de la ecuación ():

$$\overline{E}(\bar{r}) = -j\frac{\eta}{\lambda}\cdot\frac{e^{-jkr}}{r}\cdot\iint_S\left[\hat{r}\times\left(\hat{n}_S\times\overline{H}_{inc}\right)\right]\times\hat{r}\cdot e^{-jk\bar{r}'\cdot\hat{r}}\cdot dS$$

$$\overline{H}(\bar{r}) = -j\frac{1}{\lambda}\cdot\frac{e^{-jkr}}{r}\cdot\iint_S\left[\hat{r}\times\left(\hat{n}_S\times\overline{H}_{inc}\right)\right]\cdot e^{-jk\bar{r}'\cdot\hat{r}}\cdot dS \qquad \textit{Ecuación 1-39}$$

que son los campos eléctrico y magnético de la Óptica Física, y que permiten evaluar cualquier geometría arbitraria sin más que definir sus parámetros geométricos, y calcular el campo incidente en la geometría.

Anexo 1.4.4 Introducción a la Óptica Física para geométricas segmentadas

Partiendo del apartado anterior en el que se define la contribución de campo que produce en un punto cualquiera, y considerando un volumen de tamaño eléctricamente grande sobre el que incide una onda plana, debido a las elevadas dimensiones geométricas del volumen, no se tiene por que presentar una corriente inducida uniforme en todo el volumen (o la superficie, según sea), por lo que la aproximación de Óptica Física no daría un resultado correcto. Por lo tanto para resolver el problema debe recurrirse a una descomposición del volumen en trozos más pequeños, o discretización, donde cada uno de los trozos resultante es definido como faceta. Esta discretización se realiza con el fin de calcular con la mayor precisión posible la corriente inducida en distintas zonas del volumen y así asemejarse lo más posible al comportamiento real.

A partir de estas facetas, se calcula el campo en un punto cualquiera del espacio mediante las técnicas de PO, calculando la contribución de cada faceta por separado. Esto supone que para evaluar el problema, primero debe estimarse el campo incidente y a partir de él la contribución del campo en el punto considerado, una vez para cada faceta en la que se divide la estructura. Esta estrategia es evidentemente muy costosa en términos computacionales, lo que obliga a incorporar al algoritmo diferentes métodos de reducción del volumen de cómputos a realizar. Por ejemplo, resulta de gran ayuda adoptar algoritmos de evaluación de partes vistas y ocultas con el fin de eliminar del proceso de cálculo a aquellas facetas que no tendrán contribuciones significativas al valor de campo en el punto que se va a evaluar.

Anexo 1.5 Introducción a la representación mediante fuentes equivalentes

Tal y como se indica en apartados anteriores, cuando se trabaja con fuentes radiantes muy complicadas, a la hora de realizar un análisis del conjunto la complejidad matemática se dispara de tal forma que el problema se vuelve inabordable. Para conseguir solucionar estos problemas tan extensos con la menor pérdida de información posible, se debe recurrir a técnicas que generen sistemas equivalentes, que bajo ciertas consideraciones, conseguirán obtener los mismos resultados que el sistema original.

Anexo 1.5.1 Teorema de equivalencia superficial: Principio de Huygens

El teorema de equivalencia superficial, o principio de Huygens, es tal que sirve para representar fuentes reales[6] de campos electromagnéticos, mediante fuentes equivalentes,

[6] Reales: desde el punto de vista de que puedan encontrarse físicamente, tales como antenas.

que generarán idénticos niveles de campo eléctrico y magnético en una zona del espacio que rodea dicha fuente, y que será la zona de interés para el problema. Este principio de equivalencia es extensamente utilizado debido a su mayor versatilidad en el uso de aproximaciones matemáticas que si se trata con las fuentes originales.

El principio de equivalencia superficial fue introducido en 1936 S.A. Schelkunoff [30], y se basa en una formulación matemática rigurosa del principio de Huygens [31], que dice *"cada punto en un frente de onda primario, puede ser considerado como una fuente de un nuevo frente de onda esférico secundario, y a partir de esta, se puede construir un nuevo frente de onda como la suma de las envolventes de los frentes de onda generados por todas las fuentes secundarias"*. El teorema de equivalencia superficial, está basado en el teorema de unicidad, que dice *"Un campo en un medio con pérdidas, es especificado unívocamente por las fuentes que se encuentran dentro del medio, mas las componentes tangenciales del campo eléctrico sobre la frontera, o las componentes tangenciales del campo magnético sobre la frontera, o las primeras sobre una parte de la frontera, y las segundas sobre el resto de la frontera"*. Los campos en un medio sin pérdidas se consideran como el límite cuando las pérdidas tienden a cero, de los campos de un medio con pérdidas. Por lo tanto, si los campos tangenciales sobre una superficie cerrada son perfectamente conocidos, los campos en la región libre de fuentes pueden ser calculados.

A través del principio de equivalencia, se pueden definir los campos electromagnéticos que rodean una superficie cerrada imaginaria que contiene en su interior la fuente de campo original, colocando sobre dicha superficie cerrada una distribución de densidades de corriente eléctrica y magnética. Estas densidades de corriente, pueden ser calculadas sin más que definir las condiciones de contorno que rigen sobre la superficie cerrada y aplicar las ecuaciones de Maxwell. Estas densidades de corriente son escogidas de forma que el campo en el interior de la superficie cerrada es nulo, y en el exterior, el campo es igual que el campo radiado por la fuente original. Por lo tanto, es evidente que esta técnica puede utilizarse para calcular los campos radiados fuera de una superficie cerrada, que encierra las fuentes originales.

En la mayoría de las aplicaciones, la superficie cerrada se selecciona de forma que en su mayor parte coincide con las partes conductoras de la estructura física. Esta característica es debida a que las componentes de campo tangencial son evanescentes sobre las superficies conductoras (el campo eléctrico en el interior de un conductor perfecto es nulo), lo que resulta en una disminución de los límites de integración (de un volumen de integración, se pasa a una superficie).

El principio de equivalencia se desarrolla considerando una fuente situada en un medio homogéneo, y que se representa mediante una densidad de corriente eléctrica J_1 y una densidad de corriente magnética M_1. Esta fuente radia en todo el espacio, produciendo unos campos E_1 y H_1. El objetivo de la evaluación es "encerrar" esta fuente inicial de campo en una superficie cerrada S, y conseguir calcular los campos en el exterior de esta superficie, que será la región de interés. Se denota el volumen interior de la superficie por V_1 y el volumen exterior por V_2. El primer paso del desarrollo se centra en reemplazar el problema original, con otro equivalente que generará los mismos campos E_1 y H_1 en el volumen V_2 definido externamente a la superficie S. La complejidad del problema se reducirá enormemente si se escoge la superficie S de forma que los campos tangenciales en esta sean conocidos a priori.

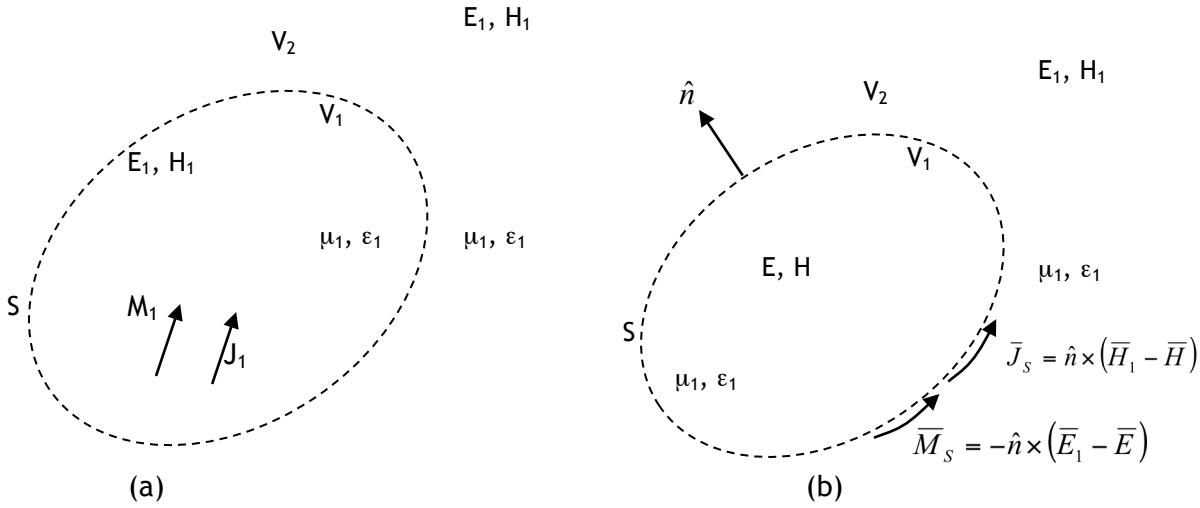

Figura 3: Teorema de equivalencia superficial

El siguiente paso del desarrollo pasa por eliminar la fuente, definida mediante J_1 y M_1, de forma que se asume que existen unos campos E y H en el interior de la superficie S, y los campos E_1 y H_1 en el exterior. Para que estos campos puedan existir, deben cumplir las condiciones de contorno para las componentes tangenciales, tanto de campo eléctrico como de campo magnético. Esto implica que, sobre la superficie S, deben existir fuentes equivalentes, de forma que cumplan que:

$$\hat{n} \times \left(\overline{H}_1 - \overline{H}\right) = \overline{J}_S \qquad \textit{Ecuación 1-40}$$

$$-\hat{n} \times \left(\overline{E}_1 - \overline{E}\right) = \overline{M}_S \qquad \textit{Ecuación 1-41}$$

Que radian en un medio indefinido y homogéneo.

En este punto, se tienen los campos originales (E_1, H_1) en el volumen V_2, y los campos (E, H) en el volumen V_1. Este volumen V_1 no es la región de interés para la resolución del problema, con lo que los campos (E, H) pueden ser cualesquiera que puedan existir y cumplir las condiciones de contorno (*Ecuación 1-40*) y (*Ecuación 1-41*), como por ejemplo ambos nulos. En esta situación, el problema equivalente se reduce a que:

$$\bar{J}_S = \hat{n} \times \left(\bar{H}_1 - \bar{H} \right)_{\bar{H}=0} = \hat{n} \times \bar{H}_1 \qquad \text{Ecuación 1-42}$$

$$\bar{M}_S = -\hat{n} \times \left(\bar{E}_1 - \bar{E} \right)_{\bar{E}=0} = -\hat{n} \times \bar{E}_1 \qquad \text{Ecuación 1-43}$$

Esta representación del principio de equivalencia es conocida como el "Principio de equivalencia de Love" [32]. Como las densidades de corriente (*Ecuación 1-42*) y (*Ecuación 1-43*) radian en un medio homogéneo, se pueden utilizar directamente las ecuaciones (*Ecuación 1-5*) a (*Ecuación 1-7*) para calcular el campo en cualquier punto.

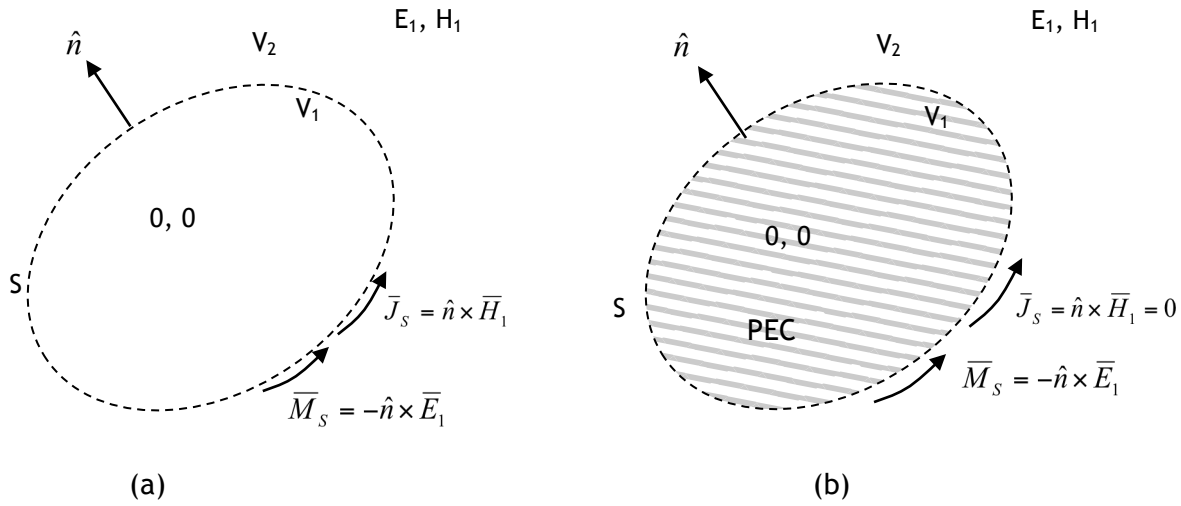

Figura 4: Principio de equivalencia de Love

Si se analiza ahora el principio de equivalencia de Love, se obtiene que las densidades de corriente (*Ecuación 1-42*) y (*Ecuación 1-43*) producen unos campos nulos en el volumen V_1, con lo que, si se cambia el medio contenido en V_1, estos campos no sufrirían ninguna modificación (ya que son nulos). Suponiendo por tanto que el medio es reemplazado por un conductor eléctrico perfecto ($\sigma = \infty$) (el problema sería simétrico para el caso de que se introdujera un conductor magnético perfecto), se producirá una condición intrínseca sobre la componente tangencial del campo magnético y en consecuencia sobre las densidades de corriente eléctrica superficiales, que se convertirán en corrientes inducidas sobre la superficie conductora, "desapareciendo" como concepto de corrientes equivalentes. Por lo

tanto, introduciendo un material PEC en el volumen V_1, quedarán en el problema únicamente las densidades de corriente magnética equivalentes de la ecuación (*Ecuación 1-4*) sobre la superficie S, que radiarán produciendo los campos originales E_1 y H_1 fuera de la superficie S (volumen V_2), que es la región de interés.

Ahora bien, dado que no se trata de un medio homogéneo, se debe estudiar la forma en la que radiaría una densidad de corriente magnética colocada sobre una superficie conductora perfecta. Para este caso, considerando una superficie conductora infinita y plana, sobre la que se coloca la densidad de corriente magnética. Según la teoría de imágenes, el problema se convierte en otro equivalente donde la densidad de corriente se coloca sobre la superficie del conductor, y con módulo el doble del módulo de la densidad de corriente inicial, radiando ahora en un medio homogéneo e infinito.

Anexo 1.6 Introducción a los algoritmos de detección de ocultamientos geométricos

La determinación de partes vistas y ocultas en una escena formada por, en general, un conjunto de placas poligonales planas en una escena 3D, o un conjunto de líneas en un entorno 2D, es una tarea que requiere normalmente procedimientos de cálculo computacional intensivo.

Típicamente esta tarea se ha abordado desde dos puntos de vista diferentes. Para comprender estos puntos de vista hay que introducir dos conceptos; espacio de objetos y espacio de imágenes.

Por **espacio de objetos** se entiende el espacio tridimensional o bidimensional real con coordenadas continuas y sin limitaciones. Trabajando en este espacio, la precisión de los cálculos no está limitada y los resultados de las operaciones pueden ser representados en una pantalla, en un plotter o en cualquier otro instrumento sin más que escalar el objeto adecuadamente.

Por **espacio de imágenes** se entiende el espacio bidimensional discreto de una pantalla, real o virtual. La precisión de las operaciones en este espacio está limitada al número de 'píxeles' de la pantalla: dos líneas se cruzan en un 'píxel' o no se cruzan (no existen los medios 'píxeles').

Como puede verse esta distinción genera dos familias de algoritmos totalmente distintos. Los algoritmos de visibilidad que trabajan en el espacio de objetos no están ligados a ningún tipo de máquinas y son capaces de generar información geométrica real acerca de la escena. Los algoritmos de visibilidad que trabajan en el espacio de imágenes sólo dan información sobre la visibilidad de los 'píxeles' de la pantalla y están ligados directamente al tipo de pantalla.

Existe además un tercer tipo de algoritmos mixtos que aprovechan las ventajas de ambos métodos (esto es, precisión en el espacio de objetos y velocidad en el espacio de imágenes). Este tipo de algoritmos se denomina de listas de prioridad. En una primera fase trabajan en el espacio de objetos y generan una lista de prioridad que no es más que una ordenación topológica de las caras de los objetos. Esta lista de prioridad establece que un elemento de la misma sólo puede ser ocultado (total o parcialmente) por los siguientes a él en la lista pero no por los anteriores. En la segunda fase, la representación, se trabaja en el espacio de imágenes utilizando el algoritmo del pintor que básicamente trata de ir pintando las caras de los objetos de la escena según el orden impuesto por la lista de

prioridad. Como puede observarse, el último proceso es lineal, lo que resulta de especial interés a la hora de dibujar en una pantalla.

Un subconjunto muy importante de los algoritmos de lista de prioridades lo constituyen aquellos que generan la lista de una manera totalmente independiente del punto de vista. Estos algoritmos son muy útiles cuando la escena es estática y la variación se da en el punto de vista. Su utilidad radica en que el proceso más costoso, numéricamente hablando, sólo se realiza una vez (generación de la lista) y la representación, que es un proceso lineal y que si depende del punto de vista, puede realizarse casi en tiempo real.

Anexo 1.6.1 Introducción a los algoritmos de árboles BSP (Binary Space Partitioning)

El algoritmo del árbol BSP, desarrollado por Fuchs, Kedem y Naylor [33], es un método extremadamente eficiente para el cálculo de las relaciones de visibilidad entre un conjunto estático de polígonos. Consta de dos etapas: en la primera se genera una estructura que es independiente del punto de vista (el árbol BSP); en la segunda se recorre dicha estructura, teniendo en cuenta el punto de vista, dando lugar a la lista BSP. La generación del árbol es un proceso de cálculo intensivo, crece con el cuadrado del número de polígonos (caso 3D) o segmentos (caso 2D), pero sólo se realiza una vez. El recorrido del árbol es un proceso lineal, crece con el número de polígonos.

El BSP es un algoritmo que opera en el espacio de objetos y sólo trabaja en el espacio de imágenes, la pantalla, a la hora de pintar las escenas.

El origen de los algoritmos BSP se encuentra en los trabajos de Schumaker [34]. La idea es la siguiente: una escena cualquiera puede ser vista como un conjunto de caras (o de aristas en una escena 2D). Si se pudiera encontrar un plano que separara totalmente un conjunto de otro, el conjunto que estuviera al mismo lado del punto de vista podría tapar al otro conjunto pero no al revés. Cada uno de estos conjuntos se dividiría recursivamente por medio de planos hasta tener el espacio parcelado de tal manera que en cada subespacio solo hubiera un polígono. Esta partición del espacio puede representarse de manera práctica como un árbol binario cuyos nodos son los planos divisores. Como la elección de planos es arbitraria, se puede realizar de tal manera que los planos divisores sean aquellos que contengan a los polígonos de la escena, de esta manera los nodos del árbol son los polígonos de la escena. La división del espacio se realiza atendiendo a la posición de los elementos respecto a la normal del polígono raíz: semiespacio frontal (frente a la normal) y semiespacio trasero (tras la normal). Así se tiene que para cada nodo del árbol aparecen dos hijos; de uno de ellos cuelga un subárbol que contiene a todos los polígonos que están

delante del nodo raíz y del otro cuelga un subárbol que contiene a todos los polígonos que están detrás del nodo raíz.

Como se puede observar, la generación del árbol es un proceso independiente del punto de vista, sólo depende de las posiciones relativas entre polígonos (delante y detrás).

Una vez calculado el árbol BSP, que no depende del punto de vista considerado, se debe generar una lista que si dependa del punto de vista, para a través de esta, poder evaluar de una forma eficiente los polígonos visibles frente a los que no lo son. Cabe comentar que estas listas son dependientes del punto de vista, pero su generación es mucho menos costosa computacionalmente hablando que la generación del árbol BSP puesto que no es necesaria una evaluación geométrica exhaustiva salvo en casos muy concretos, con lo que el uso de estas técnicas mejora en gran medida el tiempo computacional necesario para las evaluaciones geométricas de partes vistas y ocultas.

Anexo 1.7 Redes Neuronales Artificiales

Las Redes Neuronales (RNA – Red Neuronal Artificial) pueden definirse como sistemas inteligentes basados en el conocimiento. Esto quiere decir que las RNA son sistemas, en mayor o menor medida inteligentes, que se basan en una "experiencia previa", para intentar resolver cierto número de incógnitas que se les plantean. Dicho de otra forma, las RNA tratan de asemejarse al comportamiento del pensamiento humano, es decir, acumulan experiencia, que luego les sirve para resolver situaciones más o menos complicadas. Estas asimilaciones con respecto al pensamiento humano pueden hacerse dentro de las limitaciones de las que dispone una RNA, puesto que una RNA únicamente será capaz de resolver problemas infinitamente más pequeños de lo que puede ser un problema resuelto por el pensamiento humano.

Por lo tanto, una explicación bastante aproximada del funcionamiento de una Red Neuronal, está en el hecho de que una RNA es un "aproximador"[7] universal de funciones. Esto quiere decir, que ante cualquier situación matemática que pueda expresarse mediante una o varias ecuaciones, en la que un número N de parámetros, produce una variación de la ecuación para obtener un número M de nuevos parámetros, una Red Neuronal con el entrenamiento adecuado podría aproximar el comportamiento de dicha situación matemática para obtener los mismos M resultados que se obtendrían con la aplicación de la o las ecuaciones que describen el proceso.

Y por otra parte, si se dice que las RNA "aproximan" una función con un número determinado de parámetros de entrada y salida pero que podría ser obtenida analíticamente, ¿para qué su uso? Pues bien, una vez entrenada correctamente, sería fácil comprobar que los resultados de la RNA para unos parámetros de entrada dados, se obtienen de forma muchísimo más rápida que en el caso en que se aplicase la ecuación o ecuaciones pertinentes al sistema.

Anexo 1.7.1 Redes Neuronales de Base Radial

Cuando se trata con una propagación radioeléctrica en un entorno con obstáculos, para calcular los valores de campo en un punto, deben evaluarse funciones claramente no lineales surgidas de las técnicas de alta frecuencia, en este caso la Óptica geométrica, que modelen el camino seguido por los rayos desde el transmisor al receptor. Si se desea aproximar estas funciones no lineales mediante Redes Neuronales, se debe recurrir a

[7] Se denomina aproximador porque, debido a su funcionamiento, una Red Neuronal es capaz de realizar aproximaciones a cualquier función.

arquitecturas de las RNA que puedan funcionar como aproximadores universales de funciones no lineales, como pueden ser aquellas que utilizan como funciones de activación, las expansiones Volterra, los Splines, o las funciones polinomiales [35]. En este caso, y por motivos de experiencia, se han utilizado las Redes Neuronales de Base Radial (RBFN "Radial Basis Functions Networks"), que utilizan funciones base simétricas. Se puede demostrar, partiendo del teorema de Stone-Weierstras, que una RNA que utiliza estas funciones base, es un aproximador universal de funciones [36]. En las RBFN estas funciones base polinomiales y simétricas, tienen la forma de la ecuación (*Ecuación 1-44*), es decir, son las denominadas funciones gaussianas. Además, este tipo de funciones base gaussianas, son funciones locales, lo que supone que estas funciones tienen un efecto apreciable únicamente en el entorno del punto de aplicación de la función, de forma que e puede variar una de las funciones base que representan la expansión, sin modificar los resultados obtenidos por la RNA en otras zonas más alejadas de esta modificación. Desde un punto de vista teórico de la aproximación de funciones, las RBFN poseen la propiedad de "mejor aproximación uniforme", tal y como la define Chebyshev (i.e., siempre hay una RBFN que proporciona el mínimo error deseado para una función a aproximar dada) [36].

$$G(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \qquad \textit{Ecuación 1-44}$$

Por lo tanto, la aproximación de la función no lineal que se intenta modelar en la propagación electromagnética, se basará en colocar un conjunto de funciones gaussianas cuyas medias, varianzas y localizaciones dependan de los parámetros de entrada a la Red Neuronal. Un ejemplo gráfico, y muy explicativo es el mostrado en la Figura 5, donde se observa como un conjunto de funciones gaussianas combinadas linealmente, aproximan una función no lineal.

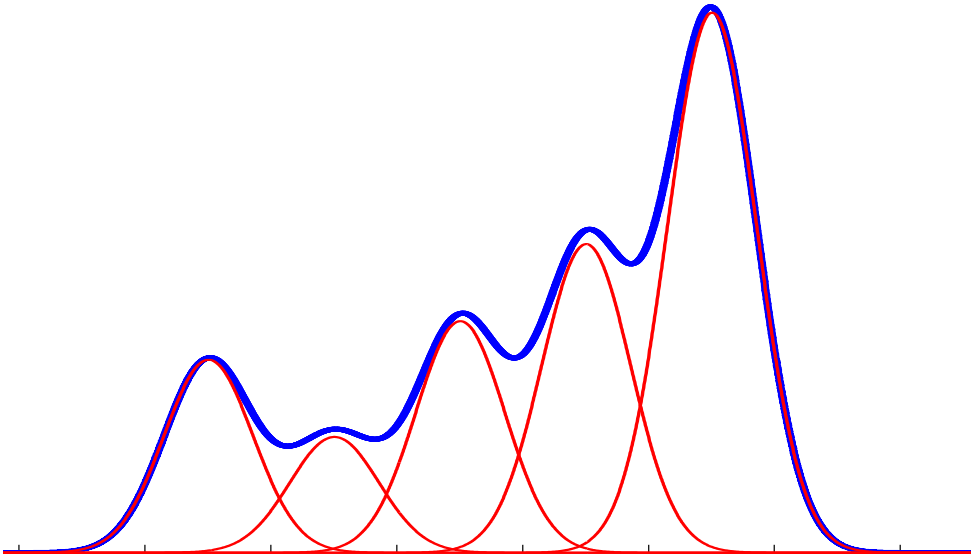

Figura 5: Aproximación de función no lineal mediante sumatorio de funciones gaussianas

Anexo 1.8 Introducción a la computación en procesadores gráficos para cálculos de carácter generalista

Con el fin de incrementar la capacidad de cómputo de los ordenadores en los últimos años, se ha comenzado a utilizar la capacidad de las unidades de procesamiento gráfico como coprocesadores de la CPU, aprovechando el gran incremento que ha tenido este tipo de dispositivos en capacidad de procesamiento.

Como se puede ver en la Figura 6, dicho aumento ha sido mucho mayor que el de las CPUs convencionales en los últimos años, las cuales se han estancado en su velocidad requiriendo utilizar varios procesadores en paralelo para aumentar su capacidad computacional.

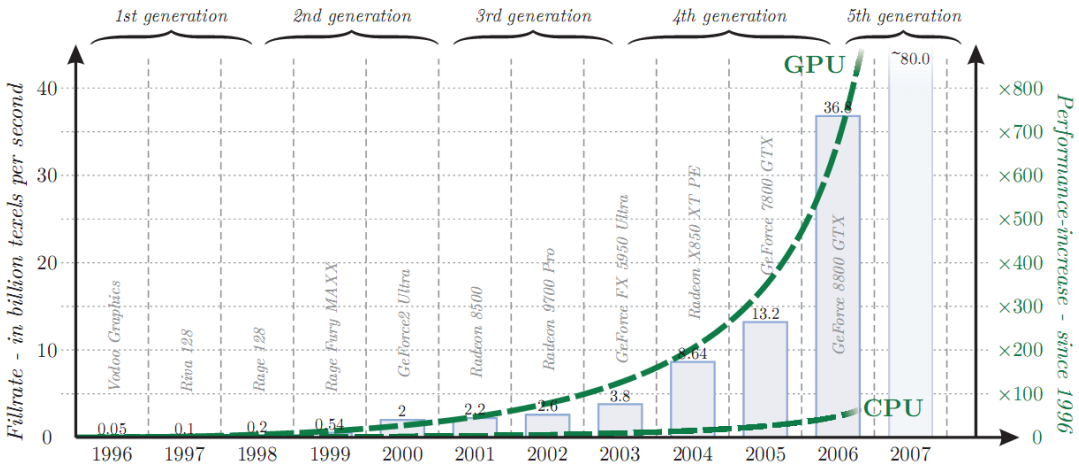

Figura 6: Comparativa del incremento de capacidad de computación entre GPUs y CPUs

En esta situación se plantea la necesidad de realizar un núcleo software que permita explotar las capacidades de cómputo de las unidades de procesamiento gráfico (GPUs) para realizar cálculos de propagación radioeléctrica sobre entornos tridimensionales completos.

Anexo 1.8.1 Programación intensiva sobre GPU

Desde hace unos años se ha extendido el uso de coprocesadores o procesadores especializados, para liberar a la CPU de ciertas tareas que pasarían a ser ejecutadas por el coprocesador en cuestión.

Un coprocesador es un microprocesador de un ordenador utilizado como suplemento de las funciones del procesador principal (la CPU). Las operaciones ejecutadas por uno de estos coprocesadores pueden ser operaciones de aritmética en coma flotante, procesamiento gráfico, procesamiento de señales, procesado de texto, criptografía, física, etc. Y su

función es evitar que el procesador principal tenga que realizar estas tareas de cómputo intensivo.

Estos coprocesadores pueden acelerar el rendimiento del sistema por el hecho de esta descarga de trabajo en el procesador principal y porque suelen ser procesadores especializados que realizan las tareas para las que están diseñados más eficientemente.

Además estos coprocesadores permiten a los compradores de ordenadores personalizar su equipamiento ya que sólo tendrán que pagar ese hardware específico quienes deseen o necesiten tener el rendimiento extra ofrecido por estos dispositivos.

Uno de estos procesadores específicos es la GPU, potente procesador de streams, capaz de sextuplicar en potencia a nuestra CPU. Hoy por hoy, cualquier ordenador personal, relativamente moderno, dispone de un procesador gráfico con una capacidad de cálculo abrumadora.

La GPU es la encargada de llevar a cabo todo el proceso de renderizado de una escena 3D, o lo que es lo mismo, dibujar en nuestro monitor todos los efectos y modelos 3D que se pueden ver cuando nos divertimos con nuestro videojuego preferido. Este proceso requiere de millones, o incluso de billones, de cálculos por segundo, lo que justifica la gran potencia que estos procesadores han de tener para poder llevar a cabo dicho proceso de renderizado.

Hace más de 25 años que se lleva investigando el modo de usar los procesadores gráficos para ejecutar tareas de propósito general. No obstante, no fue hasta el año 2000, cuando las GPUs empezaron a crecer de forma espectacular y con ellas el número de aplicaciones que aprovechaban su potencia de cálculo.

Fue en este año cuando Chris Trendall y James Stewart publicaron el documento resultante de su investigación “General calculations using graphics hardware with application to interactive caustics”.

En dicho documento se puede encontrar un amplio resumen acerca de los tipos de cálculos que las GPUs modernas soportaban en esa fecha. A partir de ese momento, fueron muchos los proyectos que empezaron a aprovechar de la potencia de la GPU para ejecutar sus algoritmos.

A continuación se presenta un breve recorrido por los conceptos, la arquitectura y la programación de propósito general sobre tarjetas gráficas (GPGPU).

Anexo 1.8.2 Introducción a las GPUs

GPU es un acrónimo utilizado para abreviar Graphics Processing Unit, que significa "Unidad de Procesamiento Gráfico". Una GPU es un procesador dedicado exclusivamente al procesamiento de gráficos, para aligerar la carga de trabajo del procesador central en aplicaciones gráficas. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la GPU, la CPU puede dedicarse a otro tipo de cálculos.

Una GPU implementa ciertas operaciones gráficas llamadas primitivas optimizadas para el procesamiento gráfico. Una de las primitivas más comunes para el procesamiento gráfico en 3D es el antialiasing, que suaviza los bordes de las figuras para darles un aspecto más realista.

Adicionalmente existen primitivas para dibujar rectángulos, triángulos, círculos y arcos. Las GPUs actualmente disponen de gran cantidad de primitivas, buscando mayor realismo en los efectos.

Anexo 1.8.3 Diferencias entre GPU y CPU

Si bien en un computador genérico no es posible reemplazar la CPU por una GPU, hoy en día las GPU son muy potentes y pueden incluso superar la frecuencia de reloj de una CPU antigua (más de 600MHz).

Pero la potencia de las GPU y su dramático ritmo de desarrollo reciente se deben a dos factores diferentes:

a. **Alta especialización de las GPUs**: ya que están pensadas para desarrollar una sola tarea siguiendo la filosofía de "stream processing" o procesamiento en cascada. En esta filosofía cada procesador modifica unos datos concretos que recibe y los transfiere al siguiente procesador, reduciendo de esta forma las necesidades de memoria caché, y permitiendo dedicar más silicio en el diseño a la parte del núcleo de proceso para llevar a cabo esa tarea más eficientemente.

b. **Alto grado de paralelismo**: Muchas aplicaciones gráficas llevan un alto grado de paralelismo inherente, al ser sus unidades fundamentales de cálculo (vértices y píxeles) completamente independientes. Por tanto, la utilización de la fuerza bruta en las GPUs es una buena estrategia para completar más cálculos en el mismo tiempo.

   Las GPU suelen tener una media docena de procesadores de vértices, y hasta dos o tres veces más procesadores de fragmentos o píxeles, aunque los modelos

actuales están cambiando al modelo unificado, que se indica en la parte final de Anexo 1.8.4.

De este modo, una frecuencia de reloj de unos 500-600MHz (el estándar hoy en día en las GPU de más potencia), muy baja en comparación con lo ofrecido por las CPU (3.8-4 GHz en los modelos más potentes), se traduce en una potencia de cálculo mucho mayor gracias a su arquitectura en paralelo.

Una de las mayores diferencias con la CPU estriba en su arquitectura. A diferencia del procesador central, que tiene una arquitectura Von Neumann, la GPU se basa en el Modelo Circulante. Éste modelo facilita el procesamiento paralelo, y la gran segmentación que posee la GPU para sus tareas.

Anexo 1.8.4 Arquitectura de la GPU

Una GPU está altamente segmentada, es decir, posee gran cantidad de unidades funcionales. Estas unidades funcionales se pueden dividir principalmente en dos: aquellas que procesan vértices, y aquellas que procesan píxeles. Por tanto, se establecen el vértice y el píxel como las principales unidades que maneja la GPU.

Adicionalmente, y no con menos importancia, se encuentra la memoria. Ésta destaca por su rapidez, y juega un papel relevante a la hora de almacenar los resultados intermedios de las operaciones y las texturas que se utilicen. Es también una de las limitaciones de la GPU dado que su tamaño suele ser pequeño y la comunicación con la memoria de la CPU, no es tan rápida como la de la GPU.

Inicialmente, a la GPU le llega la información de la CPU en forma de vértices. El primer tratamiento que reciben estos vértices se realiza en el vertex shader.

Aquí se realizan transformaciones como la traslación o la rotación de las figuras. Tras esto, se define la parte de estos vértices que se va a ver (clipping), y los vértices se transforman en píxeles mediante el proceso de rasterización.

Estas etapas no poseen una carga relevante para la GPU. Donde se encuentra el principal cuello de botella del chip gráfico es en el siguiente paso: el píxel shader. Aquí se realizan las transformaciones referentes a los píxeles, tales como la aplicación de texturas.

Cuando se ha realizado todo esto, y antes de almacenar los píxeles en la caché, se aplican algunos efectos como el antialiasing, blending y el efecto niebla.

Otras unidades funcionales llamadas ROP toman la información guardada en la caché y preparan los píxeles para su visualización. También pueden encargarse de aplicar algunos efectos. Tras esto, se almacena la salida en el frame buffer, un tipo de memoria temporal.

Hoy en día las GPU están tendiendo a un modelo unificado, esto es, procesamiento en paralelo. Los procesadores encargados de los vértices y los procesadores encargados de los píxeles, se funden bajo el nombre de Stream Processor, que aumenta la eficiencia al balancear las cargas de cada proceso (vértices y píxeles). En definitiva los procesadores que realizan el trabajo son los Shader Processor, Stream Processor se refiere a todas las unidades dentro del procesador principal.

Anexo 1.8.5 GPGPU

GPGPU, no es más que una nueva tendencia en el mundo de la programación, consistente en aprovechar la potencia que brinda el procesador gráfico para realizar tareas distintas a la de renderizado.

Estas tareas son de lo más variopintas, y entre otros muchos pueden englobarse en los siguientes ámbitos de la computación: Simulación física, Minería de datos, Visión por computador, Inteligencia artificial, Procesamiento de señales y un largo etcétera.

Anexo 1.8.5.1 Lenguajes de programación disponibles para GPGPU

A continuación se muestran los principales lenguajes de programación existentes para programación de propósito general sobre GPUs.

Anexo 1.8.5.1.1 Brook

Brook fue desarrollado por la universidad de Standford, en principio, como un lenguaje para procesadores de streaming. Posteriormente, Buck adaptó este lenguaje para convertirlo en un lenguaje de programación de propósito general sobre GPU.

Brook extiende el lenguaje de programación C con streams (una colección de elementos donde cada elemento es manipulado por las mismas instrucciones).

Los streams son diferentes a los arrays, pues no hay ningún índice de operación y no están permitidos los elementos dependientes. La funcionalidad que se aplica a cada stream se denomina *kernel*.

El proceso de desarrollo tiene dos fases: En primer lugar el programa es desarrollado y se generan ficheros de C++ que pueden ser añadidos a la compilación final.

Por desgracia, el sistema operativo de la tarjeta, el vendedor de la tarjeta y la API de la misma deben especificarse de antemano en el desarrollo. A demás desde finales de 2004 no se mantiene excepto por algunos avances para sostener *Close To Metal* (CTM) de ATI (véase Anexo 1.8.5.1.3).

Anexo 1.8.5.1.2 CUDA

El entorno *CUDA*, introducido por NVIDIA a finales de 2006, es similar a *Brook*, por ejemplo, el lenguaje de programación estándar C es extendido para soportar tipos de stream y sus operaciones correspondientes.

En contraste con *Brook*, el entorno de *CUDA* genera ejecutables completos, sin ficheros C++ intermedios. CUDA pude utilizarse como un entorno unificado para desarrollar aplicaciones para la CPU y la GPU.

Añadido al lenguaje de programación, *CUDA* también incluye librerías para algebra lineal y procesado digital de señal, que puede ser utilizado fuera del lenguaje *CUDA*.

Como desventaja, dado que es una tecnología desarrollada por NVIDIA, es aplicable, al menos por el momento, solamente a sus tarjetas gráficas.

Anexo 1.8.5.1.3 Close To Metal (CTM)

Al mismo tiempo que NVIDIA lanzaba *CUDA*, ATI introducía la plataforma CTM, una máquina virtual paralela que permite comunicación directa con los dispositivos gráficos de ATI, a través de una API gráfica.

Similar a la arquitectura CUDA, son muchas las restricciones impuestas por este enfoque, incluyendo la capacidad de leer, modificar, y escribir la memoria en un solo programa, acceder directamente a la memoria del host, o para realizar casting explícitamente entre los formatos sin copiar los datos.

CTM está distribuida como una librería que permite abrir, utilizar y cerrar conexiones gestionadas a una de las tres unidades del hardware gráfico:

1. **Procesador de comandos**: programado a través de una arquitectura independiente del lenguaje.
2. **Procesador de datos en paralelo**: programado a través de un conjunto de instrucciones nativas (arquitecturalmente dependiente).
3. **Controlador de memoria**: el cual permite acceso directo a los gráficos y la memoria principal.

Como su nombre indica, *CTM* se utiliza para acceder al hardware de gráficos en un nivel muy bajo. Se ha diseñado para optimizar los dispositivos de sintonización de la GPU basada en la funcionalidad y no para uso diario.

Además, la aplicación es responsable de todos los problemas que se producen en la programación del procesador gráfico, lo que aumenta la complejidad y los costes de desarrollo.

Anexo 1.8.6 Implementaciones de Óptica Geométrica en GPU

Una vez vistas las diferentes opciones, en lo que a métodos de cálculo de cobertura se refiere, y las ventajas y limitaciones que tiene la programación sobre tarjetas gráficas, se pasará, a continuación, a describir los diferentes tipos de implementaciones que existen de Óptica Geométrica.

No se muestran las diversas implementaciones de los otros modelos por varias razones. En primer lugar, los métodos empíricos son demasiado sencillos y para su implementación no es necesario un gran ordenador. Los métodos de onda completa, por sus requerimientos de memoria y lo lento de su ejecución, a día de hoy, no son aún una opción viable para su implementación en un PC convencional. Por otro lado de entre las dos técnicas de alta frecuencia la más ventajosa es la Óptica Geométrica, pues la Óptica Física requiere más memoria y tiempo de ejecución, no compensando aún la mejora de resultados.

Dentro de las implementaciones de óptica geométrica existen dos familias que se introducen brevemente a continuación:

a. **Ray-Tracing**: En estas implementaciones, a partir de la geometría del problema y mediante imágenes, se calculan los posibles itinerarios de la onda entre un transmisor y los puntos de recepción.

b. **Ray-Launching**: En estas implementaciones, a partir de un transmisor se envían rayos a toda la geometría estudiando su evolución dentro de la misma, hasta que su valor baja por debajo de un determinado umbral. En el avance por la misma se calcula el campo en los puntos que son atravesados por los rayos.

En la bibliografía consultada existe cierta confusión a la hora de clasificar estas técnicas, otorgándose en muchos casos el término Ray-Tracing a todas las implementaciones de GO. Esta confusión se puede ver, por ejemplo en [47] y [48]. En ambos artículos se habla de la implementación de un sistema de lanzamiento de rayos al que se nombra como Ray-Tracer.

En este documento se utilizará la clasificación antes indicada para separar ambas técnicas y así permitir la realización de un estudio en cada caso. Se puede ver un estudio más exhaustivo en [44].

Anexo 1.8.6.1 Ray Tracing

Como se puede ver en la Figura 7, esta técnica consiste en la búsqueda de posibles caminos entre un transmisor y un receptor. De esta manera, para una fuente y un receptor se obtienen las fuentes imágenes respecto de cada una de los límites de la geometría y se comprueba si hay un posible camino entre ambos.

En la Figura 7 se pueden ver algunos ejemplos de caminos: directo, simple reflexión y doble reflexión.

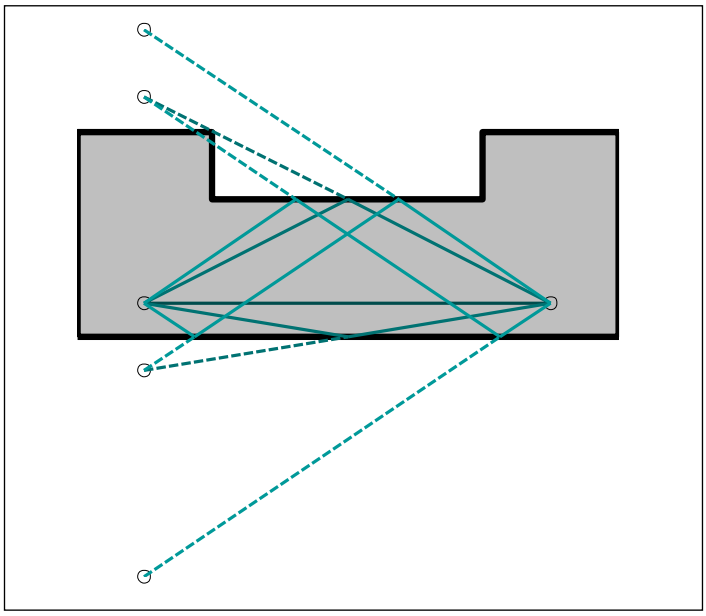

Figura 7: Ejemplo de cálculo de rayos en Ray-Tracer

Como se puede ver en [52], los tiempos de ejecución de estos algoritmos crecen exponencialmente con el número de rebotes que se permite tener al programa. Así mismo, las componentes en estudio son independientes de la energía que lleva el rayo, solamente se tiene en cuenta para la búsqueda de camino el número de rebotes permitidos.

- **Ventajas**
  a. Solamente se calculan las componentes que tienen visibilidad.
  b. Los rayos se adaptan perfectamente a la geometría, ya que la visibilidad es estudiada para el rayo concreto y no para una muestra del cono de propagación.
- **Inconvenientes**
  a. Crecimiento exponencial del tiempo de ejecución con el número de rebotes permitidos.
  b. Grandes necesidades de memoria en los casos de muchos rebotes.

Anexo 1.8.6.2 Ray Launching

La segunda familia de implementaciones de la Óptica Geométrica consiste en el lanzamiento de rayos a la geometría para el estudio del comportamiento de los mismos dentro de ésta.

De esta forma, no es necesario un estudio de la geometría para averiguar cuáles son los posibles caminos, sino que se lanzan en todas las posibles direcciones de tal forma que sea la propia evolución de los rayos la que marque los posibles caminos hasta los receptores.

Esta filosofía plantea en primer lugar un problema por la imposibilidad de lanzamiento de infinitos rayos, lo que conlleva la necesidad de lanzar rayos, no infinitesimales, sino representativos de regiones del espacio. En este sentido existen dos filosofías

a. Ángulos sólidos constantes.
b. Diferenciales de ángulo en q y f constantes.

En [54]- [55] se plantea la necesidad de utilización de ángulos sólidos constantes para que, de esta manera, la región del espacio representada por cada rayo sea similar. No obstante, en [56] se plantea la opción de la utilización de diferenciales de ángulo constante y técnicas diferentes para el cálculo de puntos contenidos en un rayo y para el análisis de estos problemas.

El lanzamiento de rayos con ángulos sólidos constantes conlleva la utilización de esferas geodésicas (Figura 8) las cuales no son fácilmente implementables. En ellas cada rayo parte de un vértice de la misma.

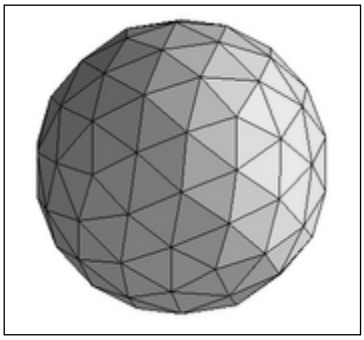

Figura 8: Esfera geodésica

Por otro lado, si se desea la utilización de diferenciales de ángulo constantes (Figura 9) su implementación es sencilla aunque se tendrán otras implicaciones.

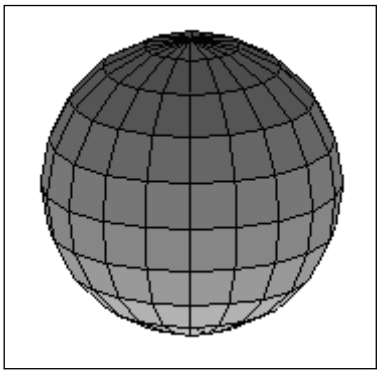

Figura 9: Esfera constante en θ y ϕ

A continuación se mostrarán las implicaciones que tiene cada una de estas técnicas, así como sus ventajas e inconvenientes. Dentro de estas ventajas se incluyen también, en ambos casos, las de Ray-Launching.

Anexo 1.8.6.2.1 Lanzamiento en esfera geodésica

La esfera geodésica o cartográfica, es una esfera definida por puntos de su superficie que equidistan a los puntos más próximos. De esta forma, si cada rayo se lanza por uno de estos puntos, dado que la distancia a los otros es constante, se pueden definir los rayos como conos.

De esta forma se pueden clasificar los puntos como pertenecientes al rayo si están contenidos en una esfera de un radio que es función de la distancia y del número de rayos lanzados (Figura 10). Puede ampliarse esta información en [48].

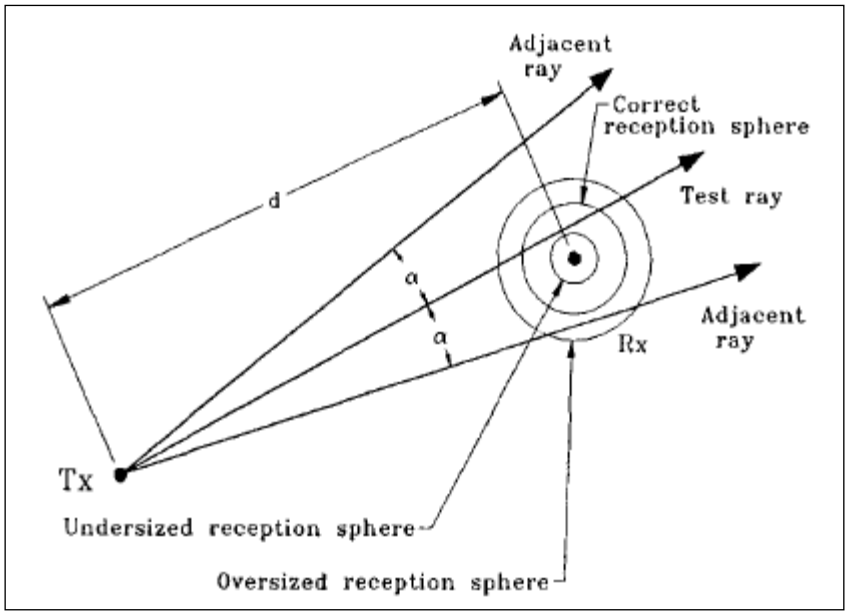

Figura 10: Definición de rayos para comprobación de punto contenido

A mayor número de vértices de la esfera, se obtiene una mayor exactitud de los rayos, pues, como se muestra en la Figura 11, con esta definición existen zonas comunes a varios rayos, lo que puede generar hasta errores de 3 dB en puntos contenidos en las mismas, o zonas de la esfera no representadas por conos.

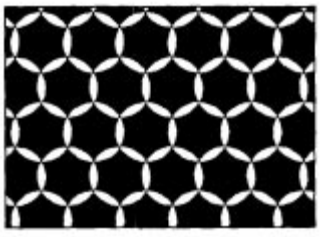

Figura 11: Errores en rayos por esfera geodésica

- **Ventajas**
  a. Simplicidad en el cálculo de puntos contenidos en un rayo.
  b. Dado que el ángulo sólido es constante, mejor reparto de los rayos en la geometría.
  c. Rayos bajo estudio dependientes de la energía que llevan, no del número de rebotes.
  d. Menor gasto de memoria.
  e. Mayor facilidad para crear iteración.
- **Inconvenientes**
  a. Complejidad en el cálculo de los rayos a lanzar.
  b. Errores por solapamiento de rayos.
  c. Lanzamiento de rayos a partes del espacio no necesarias.

Anexo 1.8.6.2.2 Lanzamiento con diferenciales de ángulo constantes

En esta técnica los rayos lanzados parten del centro de regiones de una esfera determinadas por ángulos en θ y en ϕ constantes.

Como se muestra en la Figura 12, cada rayo define una región en el espacio que no se solapa con el resto, de tal forma que la comprobación de que un punto esté contenido en un rayo consiste en el estudio de los ángulos θ y ϕ formados por el rayo y la línea transmisor-receptor.

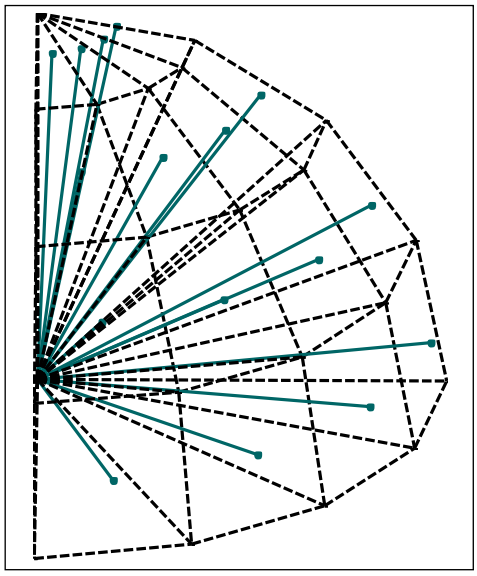

Figura 12: *Lanzamiento* de rayos mediante diferenciales de ángulo constantes

- **Ventajas**
  a. Simplicidad en el cálculo los rayos a lanzar.
  b. Rayos bajo estudio dependientes de la energía que llevan, no del número de rebotes.
  c. No existen errores por solapamiento de rayos.
  d. Menor gasto de memoria.
  e. Mayor facilidad para crear iteración.
- **Inconvenientes**
  a. Mayor complejidad en el cálculo de puntos contenidos.
  b. Dado que el ángulo sólido no es constante, los rayos se reparten de forma diferente según la elevación.
  c. Lanzamiento de rayos a partes del espacio no necesarias.

Anexo 1.8.7 Programación sobre tarjetas gráficas con CUDA

Una vez vistas en el apartado Anexo 1.8.5.1, las diferentes posibilidades de plataformas de programación para GPGPU, se presenta a continuación la situación actual sobre la más ventajosa de las mismas (CUDA).

Aunque en la actualidad comienzan a aparecer desarrollos para la utilización de CUDA, no solo desde código C++, sino también desde otros lenguajes como JAVA y dado que la versión sostenida por el fabricante es la de C++, a día de hoy es esta la opción más viable.

Por otro lado, la integración de este entorno de desarrollo en otros entornos, como puede ser Visual Studio (es el utilizado por NVIDIA para mostrar los ejemplos), es a día de hoy una labor tediosa por la cantidad de parámetros a configurar necesarios.

Es por ello que existen varias versiones de *wizards* que simplifican esta tarea y ayudan al programador en el desarrollo del código. La versión más extendida y más aceptada por los desarrolladores es la creada por *Kaiyong Zhao* (Departamento de Informática, Universidad Baptista de Hong Kong). Esta versión se puede encontrar en: http://sourceforge.net/projects/cudavswizard.

Actualmente se ha abierto como un proyecto cooperativo al que poco a poco se están añadiendo otros desarrolladores.

Anexo 2 Otras opciones de funcionamiento del simulador POBSP2D

A continuación se detallan más funciones del simulador POBSP2D

Anexo 2.1.1.1 Definición de la geometría

Para la definición de la geometría que se desea evaluar, se tienen dos opciones, bien la generación un archivo "dxf" de líneas, y su importación en el simulador, o bien la definición de la estructura directamente en el propio simulador.

Anexo 2.1.1.1.1 Creación de un fichero "dxf", e importación en el simulador

Para la creación del fichero "dxf" de entrada puede utilizarse cualquier software CAD, teniendo en cuenta las siguientes restricciones:

1. El simulador únicamente procesa las geometrías del tipo LINE y POLYLINE
2. Solamente pueden crearse estructuras cuya altura en el eje Z sea nula (en caso contrario el simulador generará un error y desechará la geometría).
3. El color de las líneas no tiene ninguna relevancia, pues el simulador no lo tiene en cuenta a la hora de realizar la importación.

Una vez realizado el fichero "dxf", este debe ser importado en el simulador. Para ello, en la barra de menú del entorno gráfico, en el menú archivo, existe el comando "Abrir DXF", que generará una nueva ventana donde se podrá definir el fichero "dxf" que se desea importar.

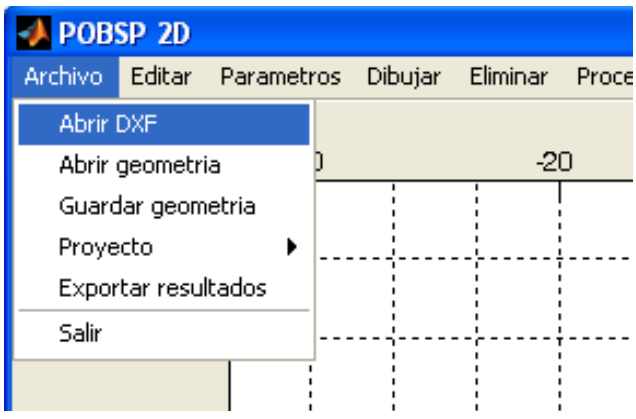

Figura 13: Importación de ficheros "dxf"

Una vez seleccionado el fichero "dxf" a importar, el simulador realizará las comprobaciones necesarias sobre el mismo, y si éste tiene la estructura y el formato correctos, solicitará al usuario la definición de los siguientes parámetros necesarios. Para

esta definición, el simulador muestra un diálogo que permite definir los parámetros electromagnéticos de las líneas, así como la definición de los vectores normales de cada línea con dos opciones diferentes, la definición manual de cada una de las normales de las líneas que definen la geometría, o la duplicación de dichas líneas y la definición de "normales opuestas"

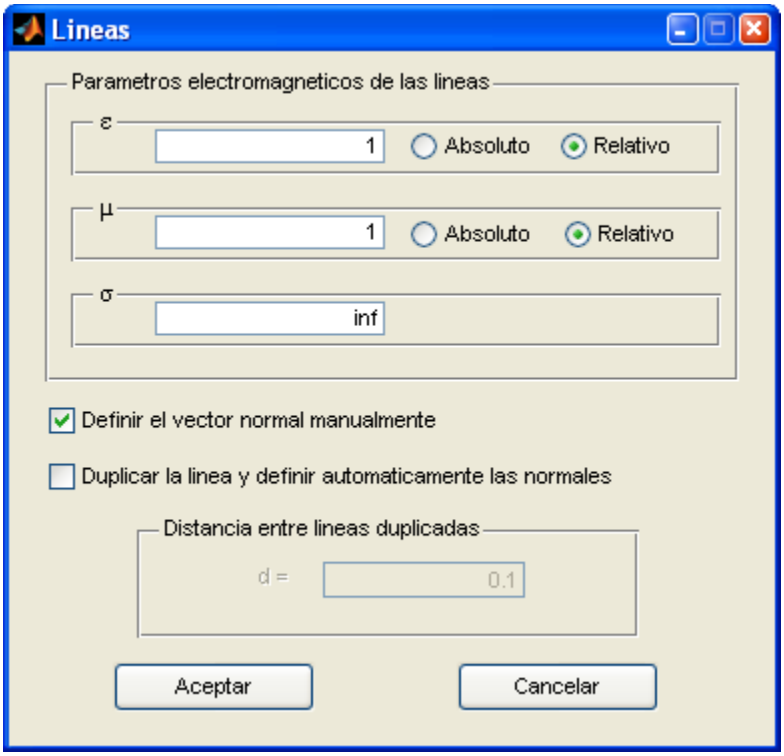

Figura 14: Definición de parámetros de las líneas

La definición manual de los vectores normales de las líneas que definen la geometría, supone que el simulador representa la estructura geométrica completa, y va solicitando al usuario la zona a la que orientar la dirección del vector normal de cada una de las líneas que definen la estructura geométrica del fichero "dxf". Para ello representa la línea para la que solicita la orientación de su vector normal, de un color distinto (color rojo) al del resto de líneas de la geometría (representadas estas en negro y verde), y el usuario debe, mediante un clic de ratón, indicar la zona hacia la que orientar la citada normal.

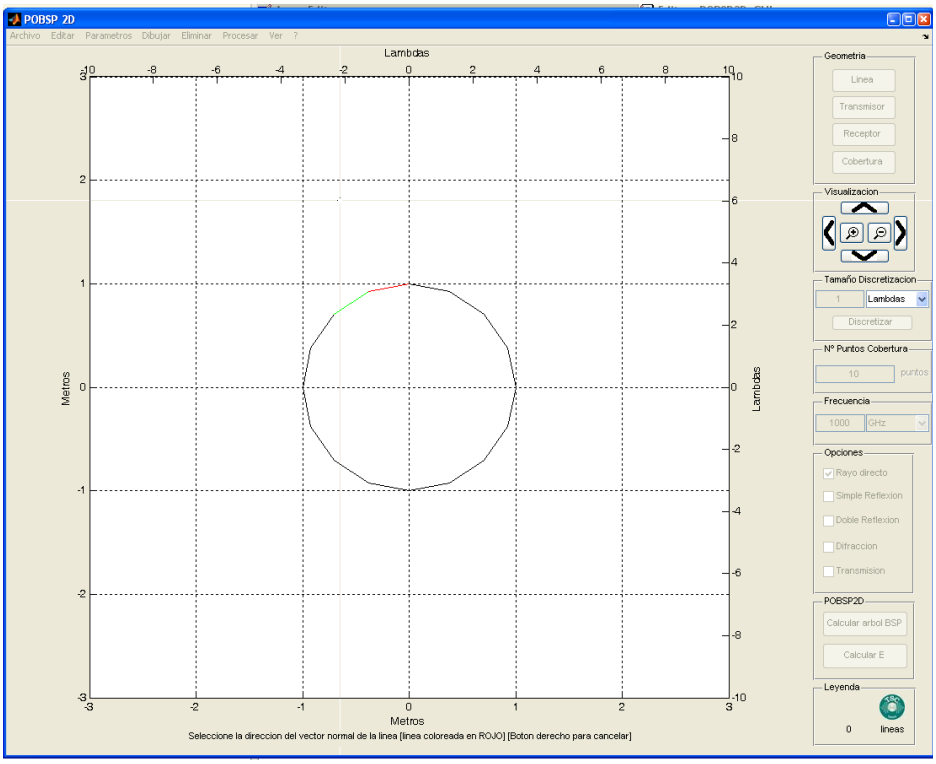

Figura 15: Definición manual de vectores normales

La duplicación de las líneas de la estructura geométrica y la definición de “normales opuestas”, supone que el propio simulador genera dos líneas por cada línea definida en el fichero “dxf” separadas estas por una distancia prefijada, y crea una normal para cada una de ellas que apunta en la dirección contraria a la otra de las líneas que se definen.

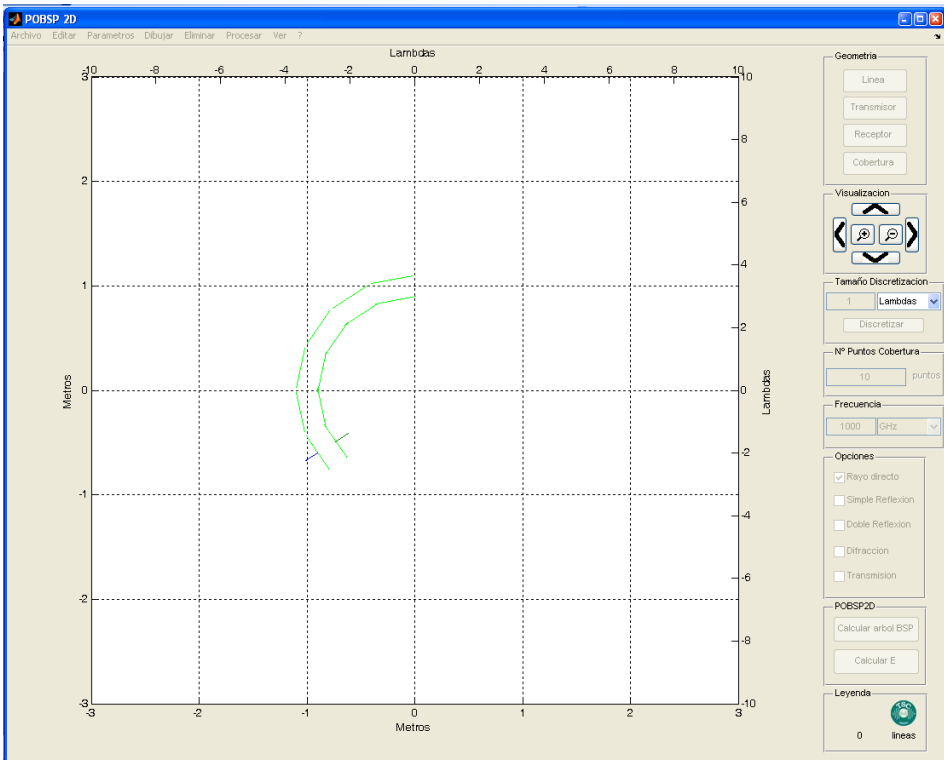

Figura 16: Duplicación automática de líneas

Anexo 2.1.1.1.2 Generación manual de la geometría

Si se desea realizar una definición manual de las líneas que componen la geometría a evaluar, debe definirse cada línea por separado a través del botón de acceso rápido del menú de la parte derecha del interfaz (Línea), o a través de la barra de menú del entorno gráfico, en la sentencia “Dibujar->línea”. La creación de esta línea supone la definición del punto inicial y del punto final de la misma, y el simulador automáticamente mostrará una ventana de definición de la línea, donde se pueden corregir los puntos inicial y final de la línea, así como la forma en la que el simulador genera los vectores normales, dando al usuario las mismas opciones que en el caso de importar un fichero “dxf”, definir los parámetros electromagnéticos del material que compone las líneas, definir manualmente el vector normal, o duplicar la línea y generar automáticamente los vectores normales.

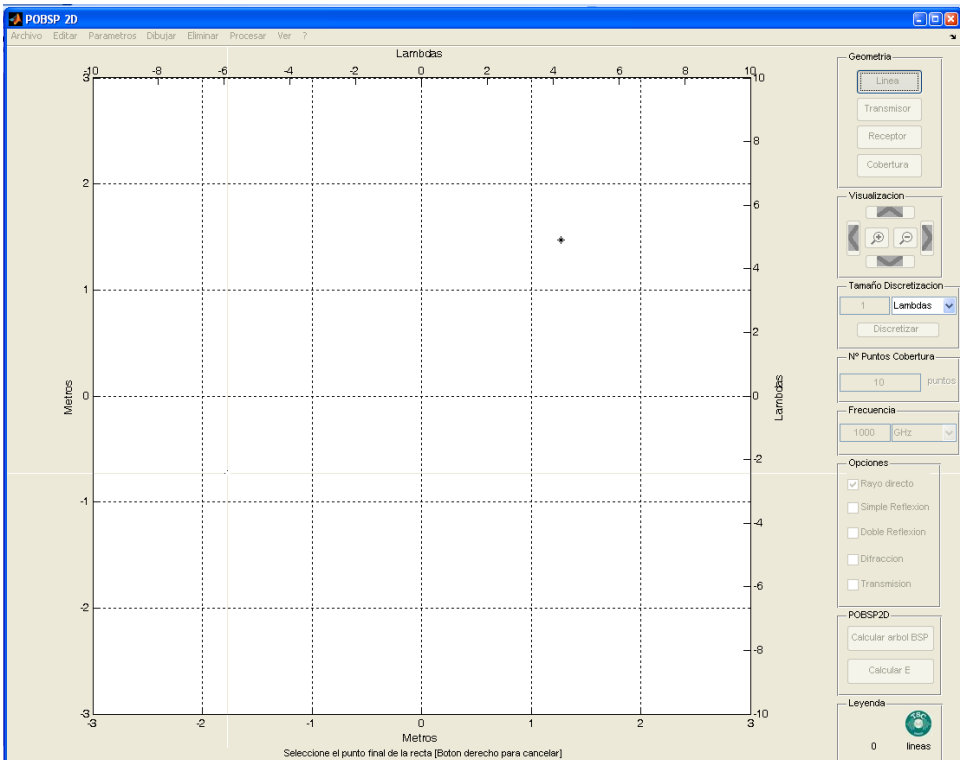

Figura 17: Definición de los puntos inicial y final de la línea

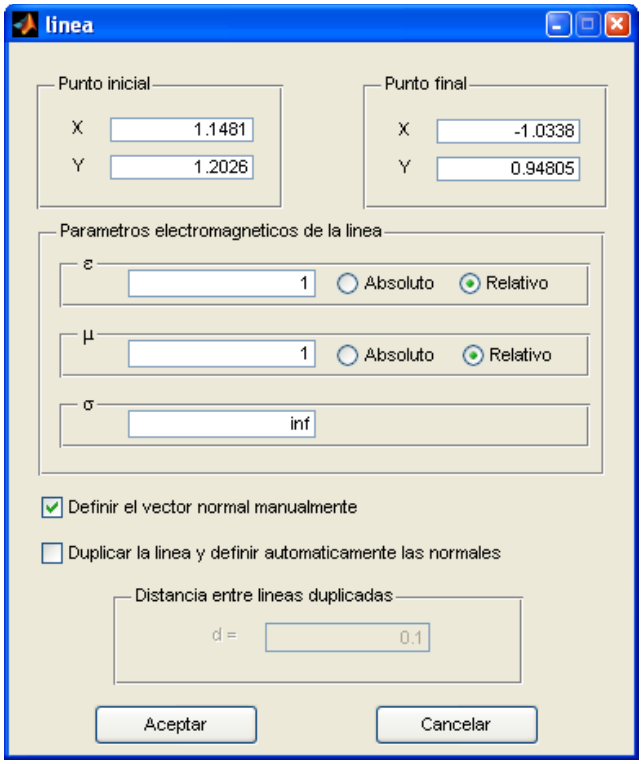

Figura 18: Coeficientes de la línea

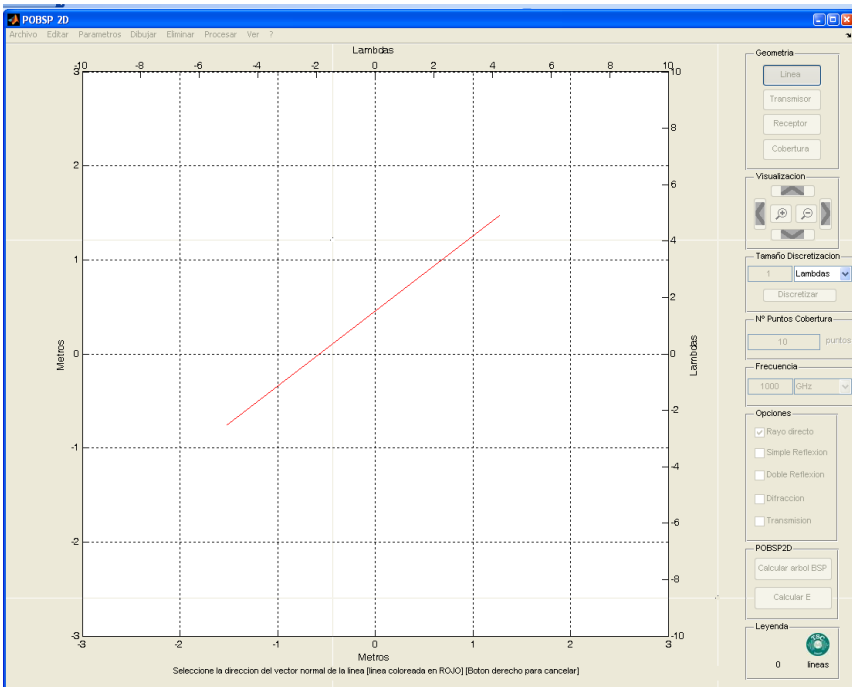

Figura 19: Definición manual del vector normal

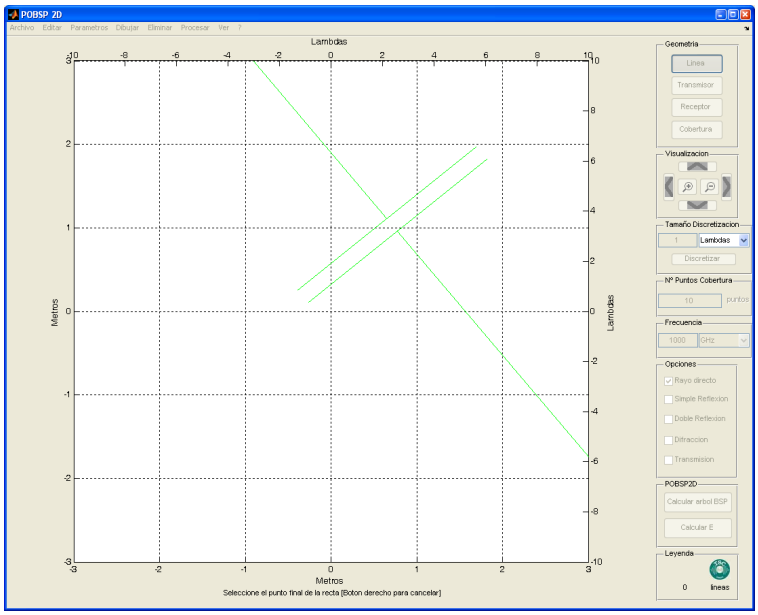

Figura 20: Definición automática del vector normal

Anexo 2.1.1.1.3 Procesado de la geometría

Una vez definida la geometría, debe ser procesada para su evaluación. Para ello, debe realizarse una discretización de las líneas que componen la estructura con el fin de que al calcular las corrientes superficiales de la evaluación por Óptica Física, éstas se ajusten adecuadamente a los valores correctos.

Para llevar a cabo esta discretización, debe definirse un tamaño máximo de las nuevas líneas, en longitudes de onda, en el cuadro de texto destinado a tal efecto en el menú de la derecha, o bien a través de la barra de menú, en la sentencia “Parámetros->Tamaño de discretización”. Una vez definido este tamaño, para proceder a la discretización se puede utilizar el botón de acceso rápido destinado a tal efecto en el menú de la derecha, o en la barra de menú, en la sentencia “Procesar->Discretizar”.

Una vez discretizada la estructura, la geometría está ya en disposición de ser procesada en los núcleos electromagnéticos para calcular los valores de campo en los puntos definidos.

Anexo 2.1.1.2 Definición de transmisor

Para la evaluación de los niveles de campo, se debe definir el transmisor que genera la excitación inicial del problema. Para ello, se puede utilizar el botón de acceso rápido destinado a tal efecto en el menú de la derecha, o a través de la barra de menú, mediante la sentencia "Dibujar->Transmisor". Se han implementado tres tipos de transmisores genéricos, un **dipolo elemental** o dipolo de Herz, un **dipolo λ/2** colocado en vertical (eje Z), y un **diagrama de radiación definido por el usuario**. La elección de uno u otro tipo de transmisor se realiza en la fase de inserción del mismo, mediante el diálogo siguiente.

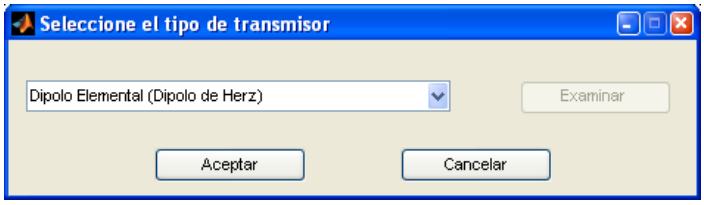

Figura 21: Definición del tipo de transmisor

Anexo 2.1.1.2.1 Definición de un dipolo elemental

Una vez seleccionado el dipolo elemental como tipo de transmisor, un diálogo solicita los parámetros de configuración del mismo, entre los que se encuentran la posición en la que colocar el dipolo, la excitación inicial que define la densidad de corriente magnética que supone el transmisor y la orientación de la misma.

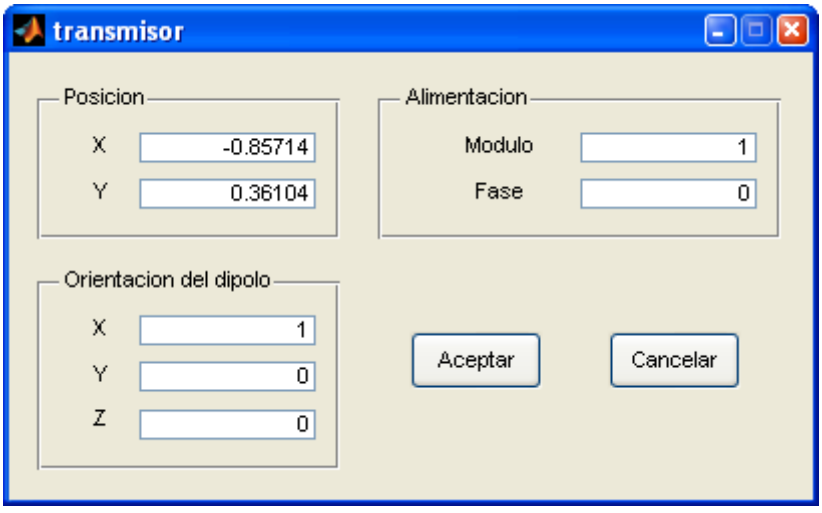

Figura 22: Definición de la excitación del transmisor

Anexo 2.1.1.2.2 Definición de un dipolo l/2 vertical

Una vez seleccionado el dipolo λ/2 como tipo de transmisor, un diálogo solicitará los parámetros de configuración del mismo, entre los que se encuentran la posición en la que colocar el dipolo y la excitación inicial.

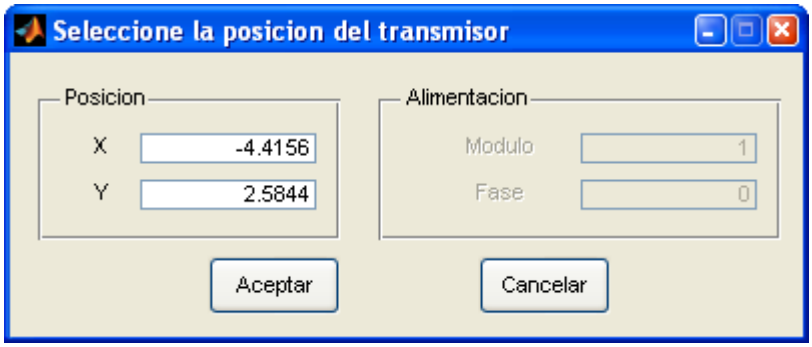

Figura 23: Definición de los parámetros del transmisor

Anexo 2.1.1.2.3 Definición de un diagrama de radiación

Cuando se desea la inserción de un diagrama de radiación definido por el usuario, se debe seleccionar esta opción en la elección del tipo de transmisor.

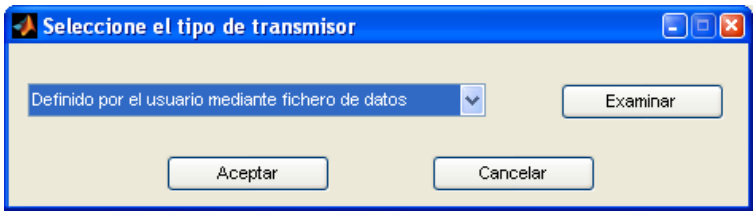

Figura 24: Definición del transmisor definido por el usuario

Una vez seleccionado este tipo de transmisor, se debe seleccionar el fichero que contiene los datos del diagrama de radiación de la antena deseada. Este fichero, almacenado en formato ASCII con tres columnas, una para el ángulo PHI (teniendo en cuenta la disposición de los ejes coordenados del simulador), la segunda para la ganancia en la coordenada PHI, y la tercera para la ganancia en la coordenada THETA.

Tras haber aceptado el transmisor, se debe definir, al igual que para el dipolo λ/2, la posición y alimentación de la antena indicada.

Anexo 2.1.1.3 Definición del receptor

Para la definición del receptor, existen dos posibilidades. Una en la que se define un único punto de recepción y en el que se obtiene por tanto un único vector de campo, y una zona de cobertura, en la que se define un número de puntos así como la zona en la que se distribuyen, y el simulador realiza una evaluación para cada uno de ellos con el fin de realizar una representación gráfica completa de los niveles de campo producidos. Para este caso, se pueden utilizar los botones de acceso rápido destinados a tal efecto en el menú de la derecha, “Receptor” o “Zona de Cobertura”, o a través de la barra de menú, mediante las sentencias “Dibujar-> Receptor” y “Dibujar->Zona de cobertura”. En el primero de los

casos únicamente se debe indicar el punto en el que colocar el receptor, y en el segundo caso se deben definir las dos esquinas que delimitan el cuadrado de la zona de cobertura, así como el número de puntos de la misma, que son solicitados por el simulador mediante una nueva ventana.

Anexo 2.1.1.4 Definición del resto de parámetros necesarios

Con respecto al comienzo de la simulación, se debe definir antes de proceder a simular la frecuencia de trabajo y las contribuciones que el simulador debe tener en cuenta a la hora de evaluar el campo en el receptor o en la zona de cobertura definida.

Para la definición de la frecuencia de trabajo, puede utilizarse el cuadro de texto definido a tal efecto en el menú de la derecha, o en la barra de menús, a través de la sentencia "Parámetros->Frecuencia de trabajo".

Para la definición de las contribuciones que el simulador debe tener en cuenta a la hora de calcular los niveles de campo, se pueden utilizar los botones seleccionables destinados a tal efecto en el menú de la derecha, o a través de la barra de menús, en las sentencias incluidas dentro del submenú "Parámetros->Reflexiones a considerar->…".

Anexo 2.1.1.5 Ejecución de la simulación

Para llevar a cabo la simulación con todos los parámetros definidos con anterioridad, puede utilizarse el botón definido a tal efecto en el menú de la derecha, o a través de la barra de menú, en la sentencia “Procesar->Calcular E”. Esto provoca el bloqueo del simulador, así como la generación de una nueva ventana que contiene una barra de estado que indica en todo momento el estado en el que se encuentra la simulación. Además, en esta nueva ventana existe un icono de cancelación, que permite abortar la simulación en cualquier instante.

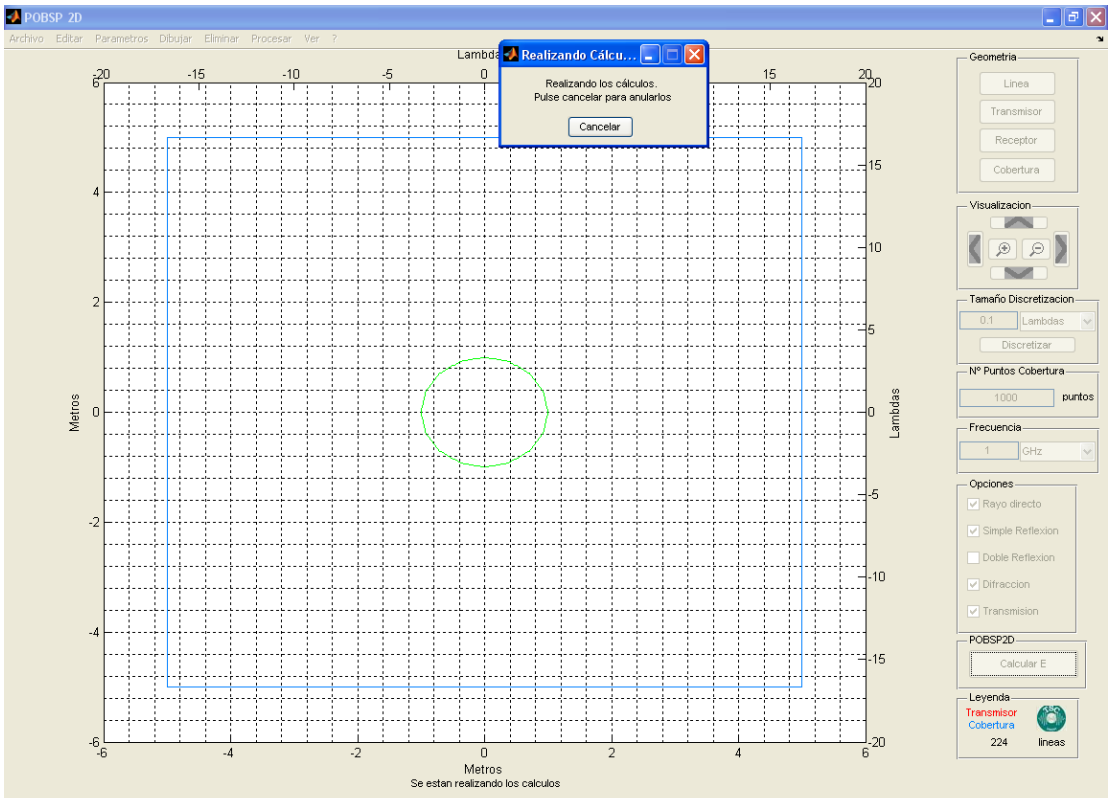

Figura 25: Evaluación de los niveles de campo

Existe la posibilidad de hacer el cálculo de la cobertura en una zona para varias frecuencias, de forma que se hace un muestreo en el número de puntos deseado en la zona indicada y se calcula el valor medio de campo para cada una de las frecuencias del muestreo. Para realizar este muestreo en frecuencia, se procede desde la barra de menú, en la sentencia “Procesar->Barrido de Frecuencias”.

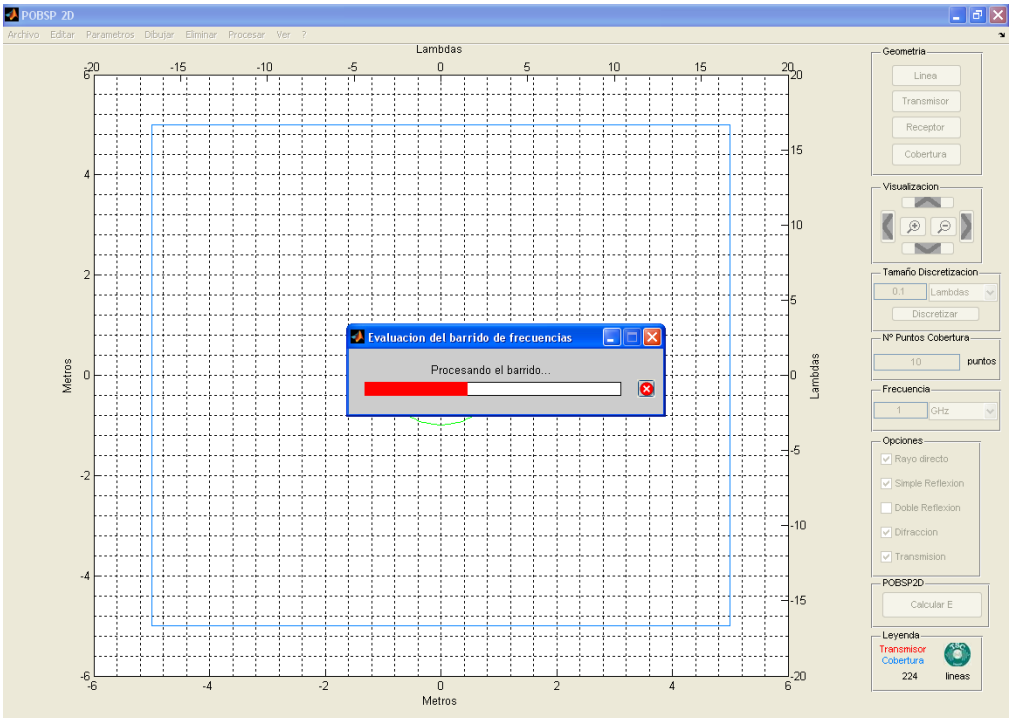

Figura 26: Barrido en frecuencia

Anexo 2.1.1.6 Presentación de los resultados obtenidos

Una vez ha finalizado la simulación con los parámetros que se han definido, el simulador genera una ventana que indica las componentes de campo eléctrico presentes en el receptor, en caso de que se haya definido una única posición para el mismo, o una serie de representaciones gráficas de las distintas componentes del campo en caso de que se haya definido una zona de cobertura.

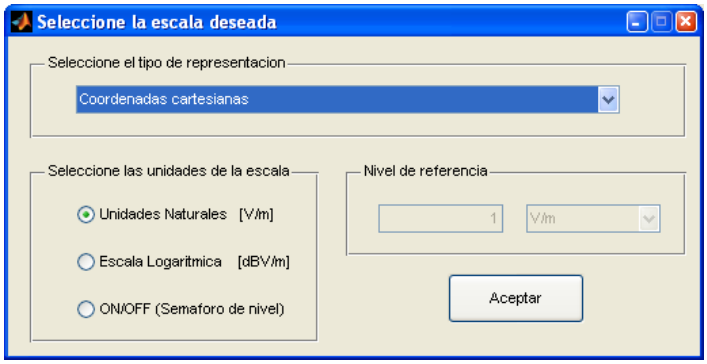

Figura 27: Selección del tipo de representación

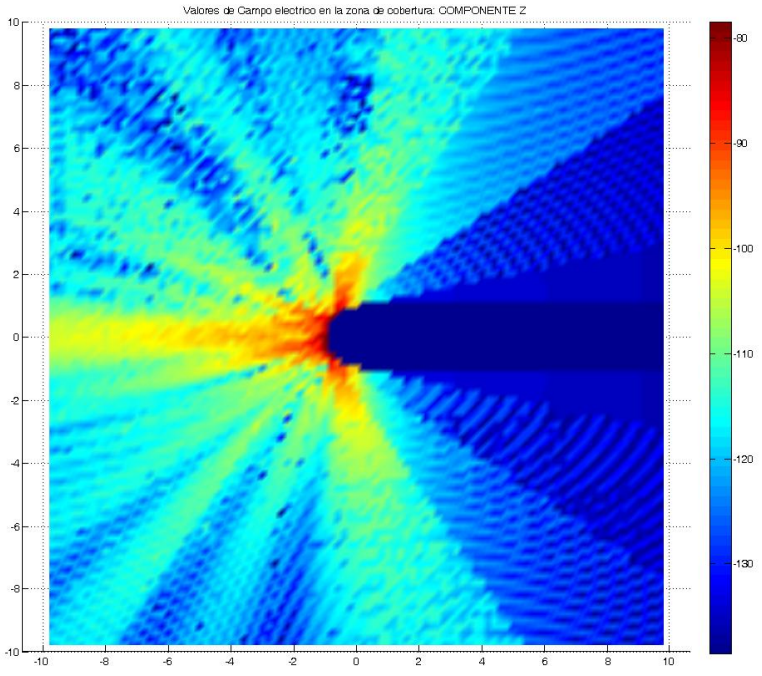

Figura 28: Resultado de una evaluación de ejemplo

En el caso del barrido en frecuencia la representación se presenta en la siguiente figura:

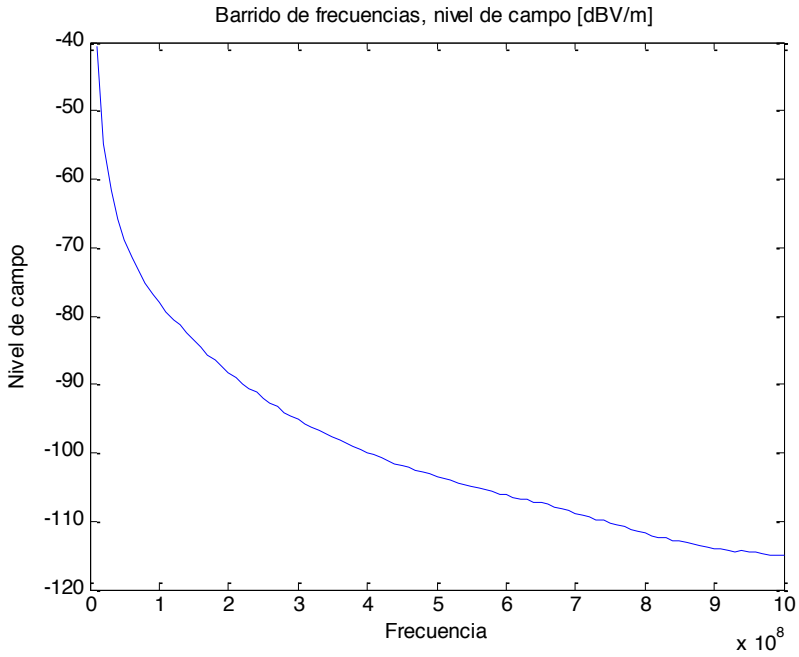

Figura 29: Representación de los resultados del barrido en frecuencia

Anexo 3 Descripción del desarrollo de la libreria GraphicOS

En los siguientes apartados se muestra una descripción de cada una de las etapas que se definen en la planificación del desarrollo.

En ellas, al mismo tiempo que se presentan las acciones a desarrollar en cada punto, se indican las clases que se modifican o crean, las pruebas que se realizan, las decisiones de implementación que se toman en ese punto o las nuevas etapas surgidas por los resultados de esa fase de desarrollo.

Anexo 3.1 Preparación de la geometría

En esta fase del desarrollo se llevan a cabo las diferentes tareas de implementación de la carga y el análisis de la geometría dentro de la tarjeta gráfica, siendo necesario para ello la lectura de la misma desde fichero, la adaptación de la información a las estructuras necesarias en la GPU y las modificaciones de la misma para optimizar las tareas posteriores.

Anexo 3.1.1 Entrada de la geometría

En primer lugar y como punto de partida se establece el fichero de entrada de la geometría. Este fichero debe permitir la descripción de geometrías tridimensionales mediante triángulos, así como diferenciar los materiales de los que están compuestos cada uno de ellos.

Una vez definidos los ficheros que recibe el programa se diseñan e implementan las clases necesarias para abstraer la obtención de los datos contenidos en ellos (*DxfReader* y *XmlReader*).

Conjuntamente con las clases indicadas anteriormente es necesaria la creación de clases auxiliares a cada una de ellas para el almacenamiento de los datos de cada uno de los ficheros. Estas clases son:

- **Triangle**: Lista dinámica de triángulos. Cada triángulo está definido por tres puntos, un índice de material y una cadena de caracteres con el nombre de la capa en la que se encuentra.
- **Material**: Lista dinámica con los materiales encontrados en el fichero *DXF*.

En las clases *XmlReader* y *DxfReader* se deben definir métodos para leer los ficheros del tipo que representan y crear las listas dinámicas necesarias para el almacenamiento de la información mediante las clases indicadas anteriormente.

En el caso de los triángulos obtenidos mediante *XmlReader*, se debe permitir, mediante la clase *Triangle*, almacenar su información en un matriz para ser guardada en un fichero (la estructura final de esta matriz se define de acuerdo a las necesidades de la GPU como se detalla más adelante).

Una vez almacenada la geometría en una lista de triángulos y otra de materiales, mediante un método de la clase *Triangle*, que recibe como parámetro la lista de materiales, se asocia la información de cada uno de los materiales a los triángulos que están definidos como pertenecientes a una capa de dicho material.

Al igual que la matriz de triángulos, se debe crear una matriz con la información de los materiales de acuerdo a las necesidades de la GPU y un vector con tantos elementos como triángulos, el cual asocia a cada triángulo un material.

Para ello es necesaria la implementación de un método en la clase *Material* que cree un vector de materiales, con un elemento más de los que tiene la lista (el primero) para incluir en la primera posición los valores de material por defecto.

Para los índices de los materiales de cada triángulo, el método se implementa dentro de la clase *Triangle*.

Anexo 3.1.1.1 Clases y estructuras creadas o modificadas

- *DxfReader*
- *XmlReader*
- *Triangle*
- *Point3D*
- *Material*

Anexo 3.1.1.2 Pruebas

- *Comprobación de la matriz de la geometría*: Para ello es necesario su almacenamiento en un fichero *CSV*.

Anexo 3.1.1.3 Decisiones de implementación

- *Fichero de la* geometría: El formato de fichero seleccionado para la geometría es *DXF*, definido por *AutoDesk*. Este formato está muy extendido actualmente en programas de representación gráfica, lo que permite que sea generado fácilmente y dota el programa de un mecanismo sencillo para la comunicación con otros programas, así como de la posibilidad de asociar

los diferentes triángulos a una capa, lo que posibilita la identificación de materiales dentro de la geometría.

- *Fichero de los materiales*: Por otro lado para indicar las características de cada uno de los materiales contenidos en el fichero de la geometría el formato seleccionado es el *XML*. De esta forma mediante un fichero *Schema* se puede definir la jerarquía del fichero a introducir.

Anexo 3.2 Abstracción de la GPU como una clase de C++

Para la comunicación con la GPU, dado que las invocaciones a la misma se realizan mediante funciones *kernel* u otras definidas dentro de la librería y con el fin de satisfacer el objetivo se abstrae la GPU y se modela como una clase de C++.

Por otro lado todos los ficheros que ejecutan kernels de *CUDA* deben estar en ficheros separados para ser compilados por el compilador de CUDA. De esta forma la abstracción debe quedar formada por tres elementos principales:

1. *Clase GPU*: Clase de C++ que abstrae de cara al programador los diferentes métodos y elementos guardados en la tarjeta gráfica. Es la encargada de la invocación de los métodos implementados en *CUDA*.
2. *Ficheros de cabecera de CUDA para la GPU*: En ellos se definen todas aquellas funciones que en su código ejecutarán *kernels* de *CUDA*.
3. *Ficheros de código de CUDA*: En el se implementan los *kernels* así como todas las funciones que invoquen dichos métodos.

Anexo 3.2.1 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.2.2 Decisiones de implementación

Para la estructuración de los ficheros de CUDA se implementan 13 ficheros diferentes:

- **Ficheros de cabecera de CUDA***:*
  - *ElectromagneticFunctions.cuh*: Definición de funciones de cálculo electromagnético.
  - *GeometricalFunctions.cuh*: Definición de funciones de cálculo geométrico.

- *GPU_functions.cuh*: Definición de funciones de propósito general para la GPU.
- *TestFunctions.cuh*: Definición de funciones implementadas para las pruebas del simulador.

- **Ficheros de kernels de *CUDA***:
  - *ElectromagneticKernels.cuh*: Núcleos de cálculo electromagnéticos.
  - *GeometricalKernels.cuh*: Núcleos de cálculo electromagnético.
  - *MatrixKernels.cuh*: Núcleos de cálculos con matrices.
  - *TestKernels.cuh*: Núcleos implementadas para las pruebas del simulador.
  - *DeviceFunctions.cuh*: Funciones a ejecutarse dentro de núcleos del simulador.

- **Ficheros de funciones de *CUDA***:
  - *ElectromagneticFunctions.cu*: Implementación de funciones de cálculo electromagnético definidas en *ElectromagneticFunctions.cuh*.
  - *GeometricalFunctions.cu*: Implementación de funciones de cálculo geométrico definidas en *GeometricalFunctions.cuh*.
  - *GPU_functions.cu*: Implementación de funciones de propósito general para la GPU definidas en *GPU_functions.cuh* e inclusión de los otros ficheros para su compilación.
  - *TestFunctions.cu*: Implementación de funciones para las pruebas del simulador definidas en *TestFunctions.cuh*.

Anexo 3.3 Envío de la geometría a la tarjeta gráfica

Una vez cargada la geometría del programa en la memoria y abstraída la GPU como una clase, se establece como siguiente paso el almacenado de dicha información en la memoria de la GPU.

A este fin, el primer paso es la obtención de una matriz que describa cada uno de los parámetros a enviar a la tarjeta (una matriz por estructura).

Para realizar los envíos se debe implementar un método dentro de los ficheros de CUDA que permita el envío de vectores, de esta forma se permite que el envío de las dos matrices y el vector se realice de la misma forma, pues las matrices son almacenadas en estructuras unidimensionales.

Las llamadas necesarias a esta función, junto con la llamada a la función que asocia los vectores a texturas se llevan a cabo desde métodos implementados en la clase *GPU*.

Anexo 3.3.1 Decisiones de implementación

Envío de la matriz de triángulos: Para el envío de la matriz de triángulos se selecciona un vector con la estructura indicada en la Tabla 1. En ella los superíndices indican el triángulo que representan y los subíndices el punto dentro del triángulo.

X_1^1	…	X_1^N	X_2^1	…	X_2^N	X_3^1	…	X_3^N
Y_1^1	…	Y_1^N	Y_2^1	…	Y_2^N	Y_3^1	…	Y_3^N
Z_1^1	…	Z_1^N	Z_2^1	…	Z_2^N	Z_3^1	…	Z_3^N

Tabla 1: Organización de la matriz de triángulos en un vector

Aunque se muestre en forma de tabla, todo ello es un vector, es decir, después de los elementos *X* están colocados los elementos *Y* y finalmente los elementos *Z*.

Envío de la matriz de materiales: En la Tabla 2 se puede ver la organización del vector de materiales que es enviado a la GPU.

ε_1	…	ε_N	μ_1	…	μ_N	σ_1	…	σ_N

Tabla 2: Organización de la matriz de materiales en un vector

Envío del vector de índices de materiales: Como último elemento a enviar, el vector de índices de materiales, es enviado mediante los mismos métodos que se implementaron para los dos casos anteriores y dado que su estructura ya es vectorial no es necesaria ninguna consideración en cuanto a su estructura.

Método para el envío de los vectores y matrices: Dado que este vector tiene elementos enteros en vez de reales como era el caso anterior, para poder utilizar los mismos métodos es necesaria la modificación de éstos para permitir cualquier tipo de vector (se utiliza el tipo void y en vez de indicarse la longitud en elementos se indica en bytes).

Anexo 3.3.2 Pruebas

Para comprobar el correcto envío de la geometría se implementan tres métodos dentro de la clase *GPU*:

- **GetTrianglesInGPU**: Este método permite recuperar la matriz de triángulos almacenada en la GPU en una nueva matriz y salvarla a un fichero para su posterior representación.
- **GetMovedGeometry**: Este método permite recuperar la matriz de triángulos almacenada en la GPU desplazada toda ella una cierta distancia en todas las direcciones.
- **GetElevatedGeometry**: Este método permite recuperar la matriz de triángulos almacenada en la GPU desplazadas las coordenadas *Z* de la misma una cierta distancia.

Con los dos primeros métodos se comprueba la disponibilidad de la información de la memoria para ser leída y obtener nuevos datos a partir de la misma.

Con el último método se permite comprobar que ésta se guarda de una forma correcta pudiéndose modificar de forma secuencial una sola de las coordenadas, en este caso la coordenada Z de los tres puntos de cada triángulo.

Anexo 3.3.3 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.4 Asociación de la geometría a texturas

Una vez almacenadas las matrices de la geometría (matriz de triángulos, matriz de materiales y vector de índices de materiales) en la memoria de la tarjeta gráfica, el siguiente paso necesario es la asociación de los mismos a texturas.

Dado que los datos de la geometría no son modificados a lo largo del programa, se puede guardar los mismos como texturas para que sean almacenados en un *buffer* y su acceso sea más rápido dentro de la tarjeta gráfica.

Para ello dentro de los ficheros de código fuente de CUDA se deben crear tres elementos del tipo texturas (texture), uno para cada una de las matrices, así como un método que permita asociar cada una de las tres matrices a cada textura.

Anexo 3.4.1 Decisiones de implementación

Como estructuras para el almacenamiento se seleccionan vectores lineales enlazados con texturas.

Anexo 3.4.2 Pruebas

Obtención de la geometría salvada en la GPU: Consiste en una modificación de la prueba para la *comprobación de la geometría cargada* para su lectura desde la textura.

Anexo 3.4.3 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.5 Creación del mallado de puntos de medición

A la hora de realizar un cálculo de cobertura sobre una geometría dada, uno de los aspectos más importantes es la elección de los puntos sobre los que se desea medir el campo, es además de gran interés que el propio programa pueda realizarlo de forma automática en base a unos ciertos parámetros introducidos por el usuario.

Un caso muy habitual es el cálculo de la cobertura una distancia por encima de la superficie de la geometría a considerar, de esta forma, si por ejemplo se desea medir cobertura móvil, se pueden considerar los puntos situados metro y medio por encima de la misma, obteniéndose una posición bastante cercana a la que podría tener un móvil en la realidad.

En base a este criterio, esta fase del desarrollo implementa un método para la creación del mallado de medición de forma automática. Para ello se siguen los siguientes pasos:

1. Obtención de los máximos y los mínimos de las coordenadas *X* e *Y*.
2. Definición de una malla de dos dimensiones con puntos equiespaciados en cada una de las direcciones en función del número de puntos deseado en cada dirección.
3. Obtención de las proyecciones de cada uno de los puntos del mallado en dos dimensiones sobre la geometría y elevación sobre la misma una distancia seleccionada por el usuario.

Para desarrollar este último punto se utilizan dos algoritmos geométricos: decisión de si un punto está contenido dentro de un triángulo en dos dimensiones (para conocer el triángulo sobre el que se debe proyectar) y la obtención de la proyección de un punto en dos

dimensiones sobre el triángulo en tres dimensiones que cruza su vertical (para la obtención de la coordenada *Z* del punto en el triángulo que lo contiene a partir de las posiciones del mallado).

Se remite al lector a los anexos de la presente memoria para analizar los algoritmos matemáticos y de mallado desarrollados.

Para la realización de esta parte del programa se crean 3 métodos dentro de los códigos CUDA: *FindMax*, *FindMin* y *CreateMesh*.

Los dos primeros permiten la obtención de los valores mínimos y máximos de una coordenada de la geometría. Explotan la ordenación de la matriz de la misma forma que se realiza en la prueba de elevación de la geometría. A partir de dichos datos y con la función *CreateMesh* se obtiene el mallado.

Es necesaria la modificación de la clase *GPU* para la inclusión de un método que realice las invocaciones de los mismos de forma ordenada y guarde los resultados (punteros a las matrices de la memoria de la tarjeta gráfica) dentro de la propia clase.

Anexo 3.5.1 Decisiones de implementación

Durante este punto del desarrollo se comprobó que si el tiempo de ejecución de un núcleo enviado a la tarjeta supera los 5 segundos, Windows considera que ésta ha entrado en un bucle infinito y genera una pantalla azul de error.

Es por este motivo que el bucle que recorre los diferentes triángulos se pasa a ejecutar fuera del núcleo. Aunque no sea la medida más eficiente, dado el tiempo consumido por esta operación (lanzamiento de cada kernel en vez de un solo lanzamiento) no se percibe apenas por el usuario, es utilizada ya que evita que el núcleo se ejecute más de 5 segundos sin devolver el control a la CPU en el caso de mallados o geometría muy grandes.

Anexo 3.5.2 Pruebas

Obtención de los puntos del mallado: Para la comprobación de la correcta creación del mallado se implementa un método en los ficheros de CUDA que permite recuperar el mallado y, al igual que las matrices recuperadas anteriormente, que en la clase *GPU* guarde en un fichero CSV los

Anexo 3.5.3 Nuevas fases creadas a partir de ésta

Centrado de la geometría: para solución de los problemas encontrados en las pruebas iniciales. Como se puede ver en la prueba, estos errores son debidos a que, dado que la

tarjeta implementa solamente reales de simple precisión (6 cifras significativas) y que la geometría real utilizada se encuentra desplazada, varias de estas cifras son utilizadas para desplazar, perdiéndose precisión en las operaciones. La solución a este problema es el centrado de la geometría.

Anexo 3.5.4 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.6 Centrado de la geometría

Dados los problemas encontrados en la creación del mallado por las cifras utilizadas para indicar la posición de cada punto, estando ésta desplazada, se crea esta fase para la realización de un centrado de la geometría y así obtener la recuperación de cifras significativas.

Como paso previo al centrado se crean tres parámetros dentro de la clase *GPU* para representar el offset, en cada una de las direcciones del espacio, que se aplica a la matriz. De esta forma es posible recuperar siempre las posiciones reales de la geometría contenida en la GPU.

Para la obtención del offset a restar en cada dirección en el centrado de la geometría, es necesario calcular los máximos y los mínimos en cada una de las coordenadas del espacio. Para ello se deben adaptar los kernels desarrollados en la creación del mallado para obtener el máximo y el mínimo, siendo utilizados para el caso de una matriz en vez de una textura.

Una vez obtenido el offset en cada dirección se implementa un kernel similar al utilizado en la prueba de la elevación de la geometría para desplazar cada una de las coordenadas de los triángulos una distancia en cada dirección.

Finalmente dentro de los ficheros de CUDA se implementa un método que a partir de los kernels anteriores obtenga el máximo y el mínimo en cada dirección, halle su valor medio y lo reste a cada una de las coordenadas.

A demás este método debe recibir por referencia las variables de offset de la clase *GPU*, quedando almacenado de esta manera el valor que se ha desplazado la geometría.

Anexo 3.6.1 Decisiones de implementación

Ejecución en la GPU: Por la gran paralelización que permite el proceso de centrado se implementa sobre la tarjeta gráfica, una vez que la matriz ha sido enviada a la misma.

Dado que en la carga de la geometría a la GPU, ésta es asociada automáticamente a una textura, se debe modificar dicho método para la realización previa del centrado, ya que una vez asociada a la textura no se pueden modificar los datos en ella contenidos (es memoria de solo lectura para asegurar que es correctamente almacenada en un *buffer*).

Anexo 3.6.2 Pruebas

Recuperación de la geometría centrada: Para la comprobación del correcto funcionamiento del centrado de la geometría se realiza la prueba en la que, a partir de los métodos implementados en pruebas anteriores para recuperar la geometría contenida en la textura, se obtiene los valores de la misma una vez realizado el centrado, comprobándose de esta manera que el centro de la misma se sitúa en el origen de coordenadas.

Anexo 3.6.3 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.7 Desarrollo del núcleo electromagnético del simulador

Con todas las clases y métodos, que permiten la carga y modificación de la geometría en el simulador, desarrollados, el siguiente paso consiste en el desarrollo de las clases y métodos que implementan la técnica asintótica de alta frecuencia GO.

A continuación se describe todo el proceso necesario para desarrollar dichos códigos, tanto aquellos métodos que permiten la creación de las estructuras de almacenamiento necesarias, como los métodos para el lanzamiento de los rayos y el cálculo del campo eléctrico, indicándose en cada paso las suposiciones que se utilizan en el desarrollo para llegar al modelo completo.

Anexo 3.7.1 Creación de estructuras de almacenamiento del campo

Como primera fase, a la hora de realizar cálculos de cobertura sobre la geometría dada, se deben crear dentro de la memoria de la GPU las estructuras necesarias para el almacenamiento de los resultados que se vayan obteniendo. Es decir, se debe crear una matriz en la que, para cada uno de los puntos, se almacene el valor de campo eléctrico.

Para la creación de dicho vector se implementa un método en los ficheros de código *CUDA* y su correspondiente llamada dentro de la clase *GPU* (*CreateFieldMatrix*), el cual, a partir de los métodos existentes en las librerías de *CUDA* y reutilizando *kernels* ya desarrollados anteriormente (en concreto el *kernel* que inicializa un vector con valores crecientes dentro de un intervalo) reserva en memoria el espacio suficiente para almacenar la matriz de campo eléctrico e inicializa el valor de todas sus posiciones a cero (se escoge como valor inicial para la asignación 0 y como incremento en los valores 0).

Anexo 3.7.1.1 Decisiones de implementación

El campo eléctrico está representado por seis valores distintos. Dado que la GPU no permite registros de tipo complejos, cada una de las componentes *(r, q, f)* vendrá representada por dos valores (su parte real y su parte imaginaria).

Partiendo de que los puntos están almacenados en una matriz bidimensional y que para cada uno de ellos se deben almacenar seis registros, la estructura lógica de almacenamiento del campo en la GPU es una matriz tridimensional.

La estructura real utilizada para la misma es un vector unidimensional en el que los elementos están almacenados como se indica en la Tabla 4.

En ella se almacenan en primer lugar las partes reales de cada uno de las componentes de campo y después las partes imaginarias.

Anexo 3.7.1.2 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.7.2 Creación y lanzamiento de los rayos

Una vez creadas las estructuras de datos en memoria, el siguiente paso es el desarrollo del código necesario para la creación de los rayos. Para ello en primer lugar los diferentes elementos necesarios para describir un rayo:

a. **Fuente del rayo**: Punto desde el que parte el rayo. En caso de ser un rayo inicial será la posición de la propia antena, en caso de ser un rebote es la posición simétrica respecto del plano de rebote de la antena.

b. **Theta**: Ángulo theta con el que el rayo parte de la antena.

c. **Phi**: Ángulo phi con el que el rayo parte de la antena.

d. **Origen del rayo**: Punto a partir del cual se considera que el rayo está activo.

e. **Final del rayo**: Punto hasta el cual el rayo está activo.

$x[0]y[0]$ $\Re e\{E_r\}$ $\Re e\{E_\theta\}$ $\Re e\{E_\phi\}$ $\Im m\{E_r\}$ $\Im m\{E_\theta\}$ $\Im m\{E_\phi\}$	$x[0]y[1]$ $\Re e\{E_r\}$ $\Re e\{E_\theta\}$ $\Re e\{E_\phi\}$ $\Im m\{E_r\}$ $\Im m\{E_\theta\}$ $\Im m\{E_\phi\}$	L	$x[0]y[N_y]$ $\Re e\{E_r\}$ $\Re e\{E_\theta\}$ $\Re e\{E_\phi\}$ $\Im m\{E_r\}$ $\Im m\{E_\theta\}$ $\Im m\{E_\phi\}$
M	M	O	M
$x[0]y[N_y]$ $\Re e\{E_r\}$ $\Re e\{E_\theta\}$ $\Re e\{E_\phi\}$ $\Im m\{E_r\}$ $\Im m\{E_\theta\}$ $\Im m\{E_\phi\}$	$x[N_x]y[1]$ $\Re e\{E_r\}$ $\Re e\{E_\theta\}$ $\Re e\{E_\phi\}$ $\Im m\{E_r\}$ $\Im m\{E_\theta\}$ $\Im m\{E_\phi\}$	L	$x[N_x]y[N_y]$ $\Re e\{E_r\}$ $\Re e\{E_\theta\}$ $\Re e\{E_\phi\}$ $\Im m\{E_r\}$ $\Im m\{E_\theta\}$ $\Im m\{E_\phi\}$

Tabla 3: Organización lógica de la matriz tridimensional de campo eléctrico

$\Re e\{E_r\}$						
$x[0]y[0]$	⊥	$x[0]y[N_y]$	⊥	$x[N_x]y[0]$	⊥	$x[N_x]y[N_y]$
$\Re e\{E_\theta\}$						
$x[0]y[0]$	⊥	$x[0]y[N_y]$	⊥	$x[N_x]y[0]$	⊥	$x[N_x]y[N_y]$
$\Re e\{E_\phi\}$						
$x[0]y[0]$	⊥	$x[0]y[N_y]$	⊥	$x[N_x]y[0]$	⊥	$x[N_x]y[N_y]$
$\Im m\{E_r\}$						
$x[0]y[0]$	⊥	$x[0]y[N_y]$	⊥	$x[N_x]y[0]$	⊥	$x[N_x]y[N_y]$
$\Im m\{E_\theta\}$						
$x[0]y[0]$	⊥	$x[0]y[N_y]$	⊥	$x[N_x]y[0]$	⊥	$x[N_x]y[N_y]$
$\Im m\{E_\phi\}$						
$x[0]y[0]$	⊥	$x[0]y[N_y]$	⊥	$x[N_x]y[0]$	⊥	$x[N_x]y[N_y]$

Tabla 4: Organización en memoria de la matriz tridimensional de campo eléctrico

Para implementar los rayos en C++ se crea una clase denominada *IDipoleRay*, la cual tiene como atributos los datos anteriormente indicados.

Con la definición de rayo establecida y la clase creada, el siguiente paso a realizar es el modelado de las antenas. Inicialmente se parte de un modelo de antena de tipo dipolo infinitesimal orientado en Z.

Así mismo, la forma de radiar al espacio se implementa mediante rayos definidos con una diferencia de ángulo entre rayos constante en θ y f.

Como datos de la misma, partiendo de estos modelos son necesarios los siguientes:

a. **Posición**: Localización de la antena en el espacio.

b. **Número de rayos en** θ: Número de ángulos diferentes que se tendrán en *q*.

c. **Número de rayos en** ϕ: Número de ángulos diferentes que se tendrán en *f*.

d. **Diferencial en** θ: Diferencia de ángulo en *q* entre dos rayos.

e. **Diferencial en** ϕ: Diferencia de ángulo en *f* entre dos rayos.

f. **Theta**: Ángulos *q* a lanzar.

g. **Phi**: Ángulos ϕ a lanzar.

h. **Frecuencia**

i. **Intensidad**

j. **Longitud**

Con estos datos se crea una clase en C++ llamada *iDipole*, la cual representa el modelo de la misma y a partir de la cual se pueden ir obteniendo los diferentes rayos a lanzar.

A si mismo, dentro de la clase *GPU* se implementa una función que, a partir de una antena dada, cree los rayos que esta genera (*ProcessIDipole*), es decir obtenga las direcciones de los rayos y, mediante la información de la geometría almacenada, calcule el final de los mismos (punto de impacto del rayo en un triángulo).

Para poder realizar los cálculos anteriormente descritos son necesarios dos *kernels* en CUDA y una función que los invoque, a demás de a otros *kernels* ya desarrollados o modificaciones de éstos. El primero de los *kernels* comprueba para cada uno de los triángulos si el rayo incide en él. Con el segundo *kernel*, para cada uno de los triángulos en

los que el rayo incide se calcula la distancia desde el origen de la antena al punto de impacto.

Una vez que se tiene la distancia a cada uno de los puntos de impacto se busca aquel triángulo que tiene una menor distancia. Para ello se implementa una modificación de la búsqueda del mínimo en la que, al mismo tiempo que busca el mínimo, devuelve el índice del vector al que corresponde dicho valor. De esta forma, a demás de conocer el punto de impacto, se puede conocer el triángulo en el que impacta.

Esta última información es útil cuando para el cálculo de la reflexión de un rayo dado que, ya que se conoce el triángulo sobre el que incide, este cálculo será directo.

Anexo 3.7.2.1 Pruebas

Puntos de impacto de rayos lanzados: para verificar que la creación de los rayos se realiza de una forma correcta se implementa la prueba, la cual, mediante rayos creados, trata de reconstruir la geometría sencilla.

Anexo 3.7.2.2 Clases y estructuras creadas o modificadas

- *IDipoleRay*
- *IDipole*
- *GPU*

Anexo 3.8 Obtención de q mínimo necesario para el lanzamiento de los rayos

Como mejora al lanzamiento de los rayos se en esta fase se implementa un método que permite calcular el valor mínimo de *q* que puede interactuar con la geometría, es decir, que puede intersecar con un triángulo o pasar por un punto.

Los rayos que tengan un valor de *q* menor que el obtenido son rayos que se lanzan hacia el espacio abierto, no aportando ninguna información, por lo que el tiempo de cómputo de los mismos es un tiempo que debe evitarse.

Cuanto más elevada esté la antena y cuanto más plana sea la geometría, mayor será la ganancia de tiempo debida a este cálculo, o mayor la ganancia en precisión para un tiempo de ejecución dado.

Anexo 3.8.1 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.9 Obtención de puntos contenidos en un rayo y cálculo del campo eléctrico generado

Con la función de procesado de la antena creada y una vez que se pueden obtener los diferentes rayos que parten de la antena, el siguiente paso es el aumento de la funcionalidad de *ProcessIDipole* para permitir calcular el campo producido por cada uno de los rayos en cada uno de los puntos.

Para ello se genera una función en el código *CUDA* que invoca dos *kernels*.

El primero de dichos *kernels* comprueba qué puntos están contenidos dentro del rayo (téngase en cuenta que cada rayo no representa solamente una línea, sino una región del espacio).

Para aquellos puntos que están contenidos dentro del rayo, el segundo *kernel* calcula el campo que la antena produce en dicho punto y lo añade a la matriz de almacenamiento del campo.

Anexo 3.9.1 Pruebas

- *Calculo del campo sobre un PEC plano e infinito.*
- *Calculo del campo sobre geometría sencilla.*
- *Calculo del campo sobre geometría real.*

Anexo 3.9.2 Nuevas fases creadas a partir de ésta

Modificación de la definición de rayo: Debida a los errores detectados, causados porque todo rallo se define mediante tres puntos: posición de la antena, origen del rayo y final del mismo. Con esta definición se considera que un punto está contenido dentro del rayo si la distancia entre él y la antena se encuentra comprendida entre las distancias del origen y el final y a demás cumple que la diferencia de ángulo con el rayo en θ y ϕ sea menor que la mitad de sus diferenciales.

En ella la zona marcada con un verde más claro es una zona que no puede ser alcanzada por el rayo pero que cumple las condiciones de ángulo y distancia. En cambio, la zona marcada en el verde más oscuro es una zona que no cumple la condición de distancia pero cuyos puntos deberían considerarse como alcanzables por el rayo.

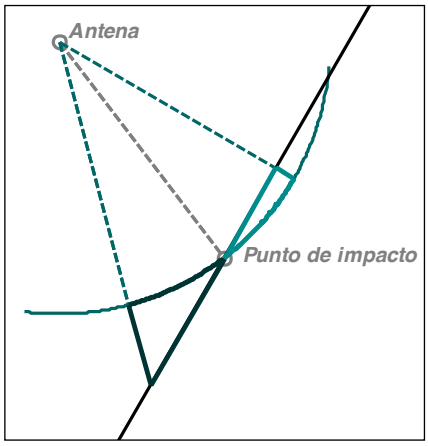

Figura 30: Descripción de los errores por la definición de rayo

Para solucionar este problema es necesaria la modificación de la definición de rayo, no considerándose los puntos iniciales y finales sino el punto origen (al igual que anteriormente) y los triángulos que delimitan su zona de propagación.

Anexo 3.9.3 Clases y estructuras creadas o modificadas

- *GPU*
- *Field*
- *Comlex*

Anexo 3.10 Modificación de la definición de rayo

Tras los errores encontrados en el desarrollo, se incluye esta fase para la modificación la definición de los rayos para indicar su origen y su final, no mediante puntos, sino mediante los triángulos correspondientes. De esta forma la definición de rayo queda definido por:

a. **Fuente del rayo**: Punto desde el que parte el rayo. En caso de ser un rayo inicial será la posición de la propia antena. En caso de ser un rebote es la posición simétrica respecto del plano de rebote de la antena.

b. **Theta**: Ángulo theta con el que el rayo parte de la antena.

c. **Phi**: Ángulo phi con el que el rayo parte de la antena.

d. **Origen del rayo**: Triángulo a partir del cual se considera que el rayo está activo (puede no haberlo y considerarse la antena).

e. **Final del rayo**: Triángulo hasta el cual el rayo está activo (puede no haberlo y considerarse que está activo hasta el infinito).

Una vez así definido el rayo se pasa a comprobar si los puntos del mallado se encontraban dentro de cada rayo mediante el análisis geométrico y no ya mediante el utilizado en el caso anterior, lo que conlleva la modificación de los métodos necesarios.

Anexo 3.10.1 Pruebas

- *Calculo del campo sobre un PEC plano e infinito.*
- *Calculo del campo sobre geometría sencilla.*
- *Calculo del campo sobre geometría real.*

Anexo 3.10.2 Clases y estructuras creadas o modificadas

- *IDipoleRay*
- *GPU*

Anexo 3.11 Generalización del dipolo infinitesimal para cualquier orientación

Una vez realizado el cálculo de la componente de campo debida al rayo directo, para permitir el cálculo del campo reflejado mediante imágenes es necesaria esta fase previa en la que se modifica la definición del dipolo, de tal forma que se permita utilizar un dipolo orientado en cualquier dirección. Esta generalización atiende a dos requisitos:

a. Permitir que la antena inicial fuese modelado en cualquier posición como dipolo.

b. Permitir crear las antenas simétricas respecto los diferentes planos para el cálculo de la componente debida a los rayos reflejados.

Para esta nueva definición de dipolo se utilizan los siguientes datos:

a. **Posición**: Localización de la antena en el espacio.

b. **Orientación**: Orientación de la corriente que recorre el dipolo infinitesimal.

c. **Número de rayos en θ**: Número de ángulos diferentes que se tendrán en θ.

d. **Número de rayos en ϕ**: Número de ángulos diferentes que se tendrán en ϕ.

e. **Diferencial en θ**: Diferencia de ángulo en *q* entre dos rayos.

f. **Diferencial en ϕ**: Diferencia de ángulo en ϕ entre dos rayos.

g. **Theta**: Ángulos θ a lanzar.

h. **Phi**: Ángulos ϕ a lanzar.

i. **Frecuencia**

j. **Intensidad**

k. **Longitud**

Con esta nueva definición del dipolo, es necesario, para el correcto cálculo del campo en cada punto, modificar la definición del rayo proveniente del dipolo para indicar en la misma la orientación del dipolo que lo ha generado. De esta forma, el rayo queda definido de la siguiente manera:

a. **Fuente del rayo**: Punto desde el que parte el rayo. En caso de ser un rayo inicial será la posición de la propia antena, en caso de ser un rebote es la posición simétrica respecto del plano de rebote de la antena.

b. **Orientación del dipolo**: Orientación del dipolo que genera el rayo.

c. **Theta**: Ángulo theta con el que el rayo parte de la antena.

d. **Phi**: Ángulo phi con el que el rayo parte de la antena.

e. **Origen del rayo**: Triángulo a partir del cual se considera que el rayo está activo (puede no haberlo y considerarse la antena).

f. **Final del rayo**: Triángulo hasta el cual el rayo está activo (puede no haberlo y considerarse que está activo hasta el infinito).

Anexo 3.11.1 Pruebas

Son las mismas del caso anterior, teniendo que mantenerse los resultados para la orientación vertical:

- *Calculo del campo sobre un PEC plano e infinito.*
- *Calculo del campo sobre geometría sencilla.*
- *Calculo del campo sobre geometría real.*

Anexo 3.11.2 Clases y estructuras creadas o modificadas

- *IDipoleRay*
- *IDipole*
- *GPU*

Anexo 3.12 Cálculo de la reflexión de cada rayo con suposición de PEC

El paso siguiente en el desarrollo del simulador consistió en dotar al mismo de la capacidad de, a partir de un rayo que incide sobre un triángulo de la geometría, calcular el rayo reflejado dentro de la misma.

Para ello, a partir del rayo inicial y del plano de incidencia, se deben calcular los nuevos datos del rayo: nuevo origen, nueva orientación del dipolo y nueva dirección del rayo.

El origen del rayo será en este caso el plano de incidencia del rayo anterior y se debe calcular el nuevo final del mismo tal y como se realiza para los rayos iniciales.

Dado que en esta fase del desarrollo se trabaja bajo la suposición de PEC, en este caso, el nuevo rayo se podrá calcular, mediante la teoría de imágenes, como proveniente de una fuente simétrica al dipolo inicial (punto y orientación simétricos respecto del plano) con una dirección tal que pase por el punto de incidencia.

Anexo 3.12.1 Pruebas

Para validar el desarrollo realizado hasta este punto se realizan una serie de pruebas que pruebas consisten, en primer lugar, en la comparación del modelo utilizado con la solución real de la teoría de imágenes para el caso de incidencia sobre un PEC plano e infinito y, en segundo lugar, en la obtención de resultados sobre la geometría compleja.

- *Comprobación de resultados con resultados teóricos en PEC plano e infinito.*
- *Comprobación de resultados con resultados teóricos en geometría real.*

Anexo 3.12.2 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.13 Lectura y utilización de diferentes tipos de diagramas

Una vez realizado el cálculo completo bajo la suposición de PEC y dipolo infinitesimal, se pasa a la implementación de la clase *Pattern,* la cual permite la lectura de ficheros de diagramas de radiación para describir la forma en la que radian las antenas.

Mediante diversos tipos de diagramas se puede reconstruir el campo radiado para así introducir las diferentes tipos de antenas dentro de la geometría.

Una vez que dichos ficheros son implementados se crean métodos (adaptaciones de los utilizados en el caso del dipolo) para obtener el campo generado por rayos definidos por un diagrama de ganancia.

Para este caso es necesario la implementación de una nueva clase para la definición de las antenas, descritas por:

a. **Posición**: Localización de la antena en el espacio.

b. **Número de rayos en θ**: Número de ángulos diferentes que se tendrán en θ.

c. **Número de rayos en ϕ**: Número de ángulos diferentes que se tendrán en ϕ.

d. **Diferencial en θ**: Diferencia de ángulo en θ entre dos rayos.

e. **Diferencial en ϕ**: Diferencia de ángulo en ϕ entre dos rayos.

l. **Theta**: Ángulos θ a lanzar.

m. **Phi**: Ángulos ϕ a lanzar.

f. **Frecuencia**

g. **Diagrama**: Diagrama de radiación de la antena.

Así como la definición de un rayo para la antena (*AntennaRay*):

a. **Fuente del rayo**: Punto desde el que parte el rayo. En caso de ser un rayo inicial será la posición de la propia antena, en caso de ser un rebote es la posición simétrica respecto del plano de rebote de la antena.

b. **Theta**: Ángulo theta con el que el rayo parte de la antena.

c. **Phi**: Ángulo phi con el que el rayo parte de la antena.

d. **Origen del rayo**: Triángulo a partir del cual se considera que el rayo está activo (puede no haberlo y considerarse la antena).

e. **Final del rayo**: Triángulo hasta el cual el rayo está activo (puede no haberlo y considerarse que está activo hasta el infinito).

f. **Ponderación del rayo**: Amplitud de la onda a un metro de la fuente del rayo, a partir de este dato se puede reconstruir toda la onda.

Para la comprobación de la correcta implementación de la lectura y utilización de diagramas se llevan a cabo dos acciones: creación de ficheros de diagramas para el dipolo elemental y cálculo del campo con dichos ficheros.

Para la primera de las acciones, a partir de la formulación del campo lejano para el caso del dipolo elemental, se generan varios ficheros que definiesen su comportamiento.

Anexo 3.13.1 Pruebas

- *Comprobación de resultados con resultados teóricos en PEC plano e infinito.*
- *Comprobación de resultados con resultados teóricos en geometría real.*

Anexo 3.13.2 Clases y estructuras creadas o modificadas

- *Antenna*
- *AntennaRay*
- *Pattern*
- *KindOfPattern*
- *GPU*

Anexo 3.14 Generalización de la reflexión para todo tipo de materiales y orientación del plano de reflexión

Como último paso, en lo que a descripción electromagnética se refiere dentro del simulador, se establece la eliminación de la suposición de PEC, para lo cual es necesario generalizar la reflexión para incidencia oblicua sobre superficies arbitrarias de características electromagnéticas variables. Se puede ver toda la formulación que conlleva esta generalización, así como los cambios de base que son necesarios en los apéndices de la presente memoria.

Para ello es necesaria la implementación de códigos dentro de la clase GPU que permitan el cálculo del coeficiente de reflexión a partir de la información electromagnética del triángulo sobre el que se incide.

Anexo 3.14.1 Pruebas

- *Simulación de PEC*: para la comprobación de la correcta generalización, se realizan las pruebas realizadas en los casos anteriores utilizando un valor de conductividad suficientemente alto (simulación de PEC), para comparar los resultados con los de PEC.
- *Simulación de dieléctrico con pérdidas*: son pruebas para ver el comportamiento en el caso de un dieléctrico y comparar con los resultados de PEC, observándose las diferencias.

Anexo 3.14.2 Clases y estructuras creadas o modificadas

- *GPU*

Anexo 3.15 Creación de la DLL y generación del programa de pruebas

Aunque se indique al final del desarrollo, la creación de la dll y algunas de las medidas necesarias para la generación de la misma son llevadas a cabo a lo largo del desarrollo del simulador y no solamente al final.

Este es el caso de la clase de error, un elemento del simulador que permite al programador conocer la causa de todos y cada uno de los errores que pueden ser causados por el núcleo software. Es por ello que se desarrolla a lo largo de todo el proyecto.

Por otro lado, por simplicidad en el desarrollo y para permitir una mayor facilidad de depuración, el proyecto inicialmente se estructura como una aplicación de consola Win32, realizándose todas las librerías necesarias, para posteriormente dividir el proyecto en dos: por un lado el proyecto que genera la dll y contiene todos los códigos de la librería y por otro un ejecutable que utilice las diferentes funciones y clases implementadas en la misma (la parte ejecutable del proyecto inicial).

A continuación se pasarán a describir cada una de estas dos labores.

Anexo 3.16 Creación de la clase de error

Dada la naturaleza de la dll, para permitir al programador el tratamiento de los diferentes errores y conocer la causa de los mismos en el desarrollo del programa, se implementa una enumeración de los diferentes errores posibles (*GraphicosError*), así como un método que permite, a partir de uno de los elementos de la enumeración, recuperar una cadena de caracteres que explique la naturaleza del error.

Así mismo, a todas las clases que, aparte de los métodos que devuelven elementos del tipo *GraphicosError*, poseen funciones que puedan devolver identificadores de error mediante otros tipos (punteros nulos, valores negativos...), se les añade un atributo que identifique el último error que acontece en la misma, para que se pueda averiguar la causa de los errores en dichas funciones.

Anexo 3.16.1 Clases y estructuras creadas o modificadas

- *GraphicosError*

Anexo 3.17 Eliminación de código utilizado para las fases de desarrollo y creación del programa de prueba de la dll

Finalmente, para dar al proyecto la forma correcta se separa en dos proyectos el código desarrollado hasta el momento: por un lado un proyecto con el código que llama los diferentes métodos implementados (fichero *main.cpp*) y otro proyecto con todas las funciones implementadas y con las diferentes clases, el cual en vez de generar un ejecutable genera una dll que es utilizada por le ejecutable de prueba.

Así mismo, se eliminan todas aquellas instrucciones utilizadas para mostrar información por pantalla de la librería durante la creación de la misma, dejando de este modo toda la labor de depuración al programador final de tal forma que puede hacer uso de la estructura *GraphicosError* para obtener toda la información.