ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Análisis de tiempos de computación de modelos de propagación radioeléctricos para la planificación eficiente de sistemas de radiocomunicaciones en la banda de UHF

AUTOR: David Pérez Piqueras
DIRECTOR: Leandro Juan Llácer

Julio / 2007

Autor	David Pérez Piqueras
E-mail del Autor	davidperpiq@hotmail.com
Director	Leandro Juan Llácer
E-mail del Director	leandro.juan@upct.es

Título del PFC	Análisis de tiempos de computación de modelos de propagación radioeléctricos para la planificación eficiente de sistemas de radiocomunicaciones en la banda de UHF

Resumen

El proyecto se centra en optimizar los tiempos de cálculo computacional que presenta la herramienta RADIOGIS (desarrollada por el departamento de Tecnologías de la Información y las Comunicaciones de la Universidad Politécnica de Cartagena) durante la realización de prácticas relacionadas con la cobertura radioeléctrica de sistemas de radiocomunicación.

Para ello, se deberá modificar el funcionamiento de los diferentes ficheros programados en Borland C++, sobre los cuales se soporta RADIOGIS para llevar a cabo los cálculos mencionados.

Finalmente, habrá que determinar si se consigue una mejora de los tiempos de cálculo, y sopesar si los posibles errores que esto implica, son lo suficientemente pequeños para que el trabajo realizado en este proyecto pueda formar parte de la aplicación definitiva de RADIOGIS.

Titulación	Ingeniero de Telecomunicación
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Julio – 2007

Agradecimientos.

Los primeros que merecen una mención especial son **mi familia**, los cuales me han apoyado incondicionalmente durante todos estos años de estudios. Gracias a mis padres Ginés y Reme, a mi hermano Adrián, y a mi novia Conchi.

Doy las gracias a mi tutor, Leandro Juan Llácer, por darme la oportunidad de realizar un proyecto teniéndolo a él como director. Ésta era mi idea desde el momento en que ingresé en la UPCT. Para mí ha sido un honor.

Importante ha sido también la ayuda que me ha prestado Rubén Ibernón, quien me ayudó en momentos en los que la materia me generaba muchas dudas

Gracias a mis familiares y amigos; la gente que me hace sentir querido.

Para finalizar, agradecer también a los profesores la formación que nos han facilitado en todos estos cursos.

GRACIAS A TODOS.

Índice.

INTRODUCCIÓN

1. INTRODUCCIÓN.

1.1. OBJETIVO DEL PROYECTO.

El proyecto gira en torno a la optimización de la aplicación *RADIOGIS*, la cual ha sido desarrollada con la ayuda de *ArcGIS Engine 9 de ESRI* para la realización de prácticas relacionadas con la cobertura radioeléctrica de sistemas de radiocomunicación.

El objetivo es aumentar la velocidad de dicha herramienta en el cálculo computacional predictivo de pérdidas, entre un transmisor y un receptor, en una determinada área de cobertura.

1.2. RESUMEN DEL PROYECTO.

Para conseguir el objetivo expuesto anteriormente, se han seguido una serie de actuaciones que se pueden dividir en seis fases:

- *FASE PRIMERA:*

Estudio teórico de diferentes modelos utilizados en transmisión por radio para la predicción de perdidas.

Concretamente se han estudiado tres modelos; un primero empleado para entornos rurales, el cual se denomina UITR-526, y otros dos útiles para entornos urbanos: COST-231 y Xia-Bertoni.

Este primer paso es indispensable para poder tener una idea de la realidad en la que se basan los diferentes archivos sobre los que posteriormente se trabajará.

- *FASE SEGUNDA:*

Realizar un estudio exhaustivo de los ficheros proporcionados por el tutor para trabajar sobre ellos, en los cuales quedan plasmados los diferentes modelos de predicción previamente mencionados. Estos ficheros, base sobre la que trabaja *RADIOGIS*, están programados en Borland C++, por lo que es necesario refrescar los conocimientos que se posean de C++, tomándolos como base para asimilar las pequeñas diferencias y grandes similitudes que este lenguaje guarda respecto a Borland C++.

El modelo que se va a analizar con más detalle es el UITR-526. Es necesario conocer todas y cada una de las líneas de código, para así poder saber cuáles son los elementos o bloques de código que se necesitan modificar para conseguir la optimización de velocidad deseada.

Los programas trabajan sobre un fichero de texto que presenta mediante alturas un modelo digital del terreno, o un mapa de pérdidas adicionales, con formato ASCII. El fichero tiene el siguiente aspecto (**ejemplo explicativo 1**):

```
                ncols         10
                nrows         10
              xllcorner     3650672
              yllcorner     5311282
                cellsize      200
              NODATA_value  -9999
460 460 460 460 460 457 449 442 440 439
446 426 418 418 412 399 384 380 380 377
368 367 367 368 375 362 371 379 371 382
397 409 416 400 395 383 387 399 408 409
411 415 417 420 411 386 375 360 360 359
351 343 340 340 340 340 332 322 320 320
320 329 339 340 347 359 364 385 395 408
418 436 457 473 471 465 457 453 449 455
454 449 447 434 430 410 396 394 384 381
364 342 333 330 328 321 320 320 335 395
```

Figura 1.1. Ejemplo explicativo 1: modelo digital del terreno en formato ASCII.

Donde:

- *ncols* es el número de columnas de la matriz.
- *nrows* es el número de filas de la matriz.
- *xllcorner* es la coordenada "x" de la esquina inferior izquierda de la matriz.
- *yllcorner* es la coordenada "y" de la esquina inferior izquierda de la matriz.
- *cellsize* es la distancia entre celdas adyacentes.
- *NODATA_value* indica el valor que tomará una celda que no tiene datos.

En el caso de un modelo digital del terreno (MDT), el valor de la celda se corresponde a la altura de dicho terreno. Si se trata de un mapa de pérdidas adicionales, el valor de la celda será las pérdidas adicionales en decibelios que se sumarán a las de propagación.

- *FASE TERCERA:*

Conocido el fichero UITR-526 a la perfección, se procede al análisis de las posibles soluciones a adoptar para que cambie su funcionamiento. La clave está en **modificar el procedimiento de levantamiento del perfil**. **El fichero original levanta el perfil linealmente; hay que intentar que lo haga en diagonal**.

Se va a hacer un pequeño resumen de lo comentado (aunque donde se explicará el cambio con más detalle es el apartado 1.4. del proyecto).

Supóngase que se dispone del MDT anteriormente mostrado, y el transmisor y área de cobertura a calcular a calcular son los siguientes:

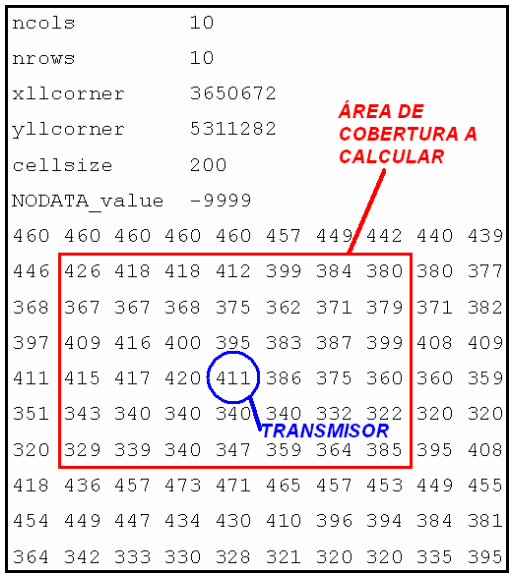


Figura 1.2. Ejemplo explicativo 1. Situación del transmisor y el área de cobertura.

El programa "recorre" la primera fila, preguntando en cada columna si se desea calcular las pérdidas de la celda correspondiente. En este caso, todas las respuestas serían negativas, puesto que ninguna celda pertenece al área de cobertura que se desea conocer.

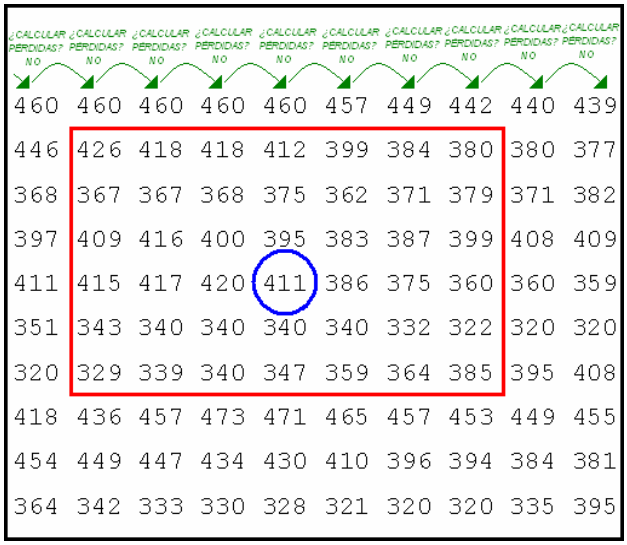


Figura 1.3. Ejemplo explicativo 1. Funcionamiento del programa original y modificado en la fila 0 del MDT.

Posteriormente, comienza su recorrido por la segunda fila. En la celda de 426 metros de altura se desea calcular las pérdidas, por lo que se procede a levantar el perfil (PERFIL 0) mediante interpolación, y tras ello se llama a un método del fichero que calculará las pérdidas sobre él.

Este procedimiento se repetirá para todas y cada una de las celdas que componen el área de cobertura (41 en este caso, porque como es natural, por incongruencia no se leventará perfil en la celda del transmisor).

Cuando se hayan recorrido el total de las celdas de la matriz del MDT, el cálculo de la cobertura habrá finalizado, y ésta será representada en pantalla.

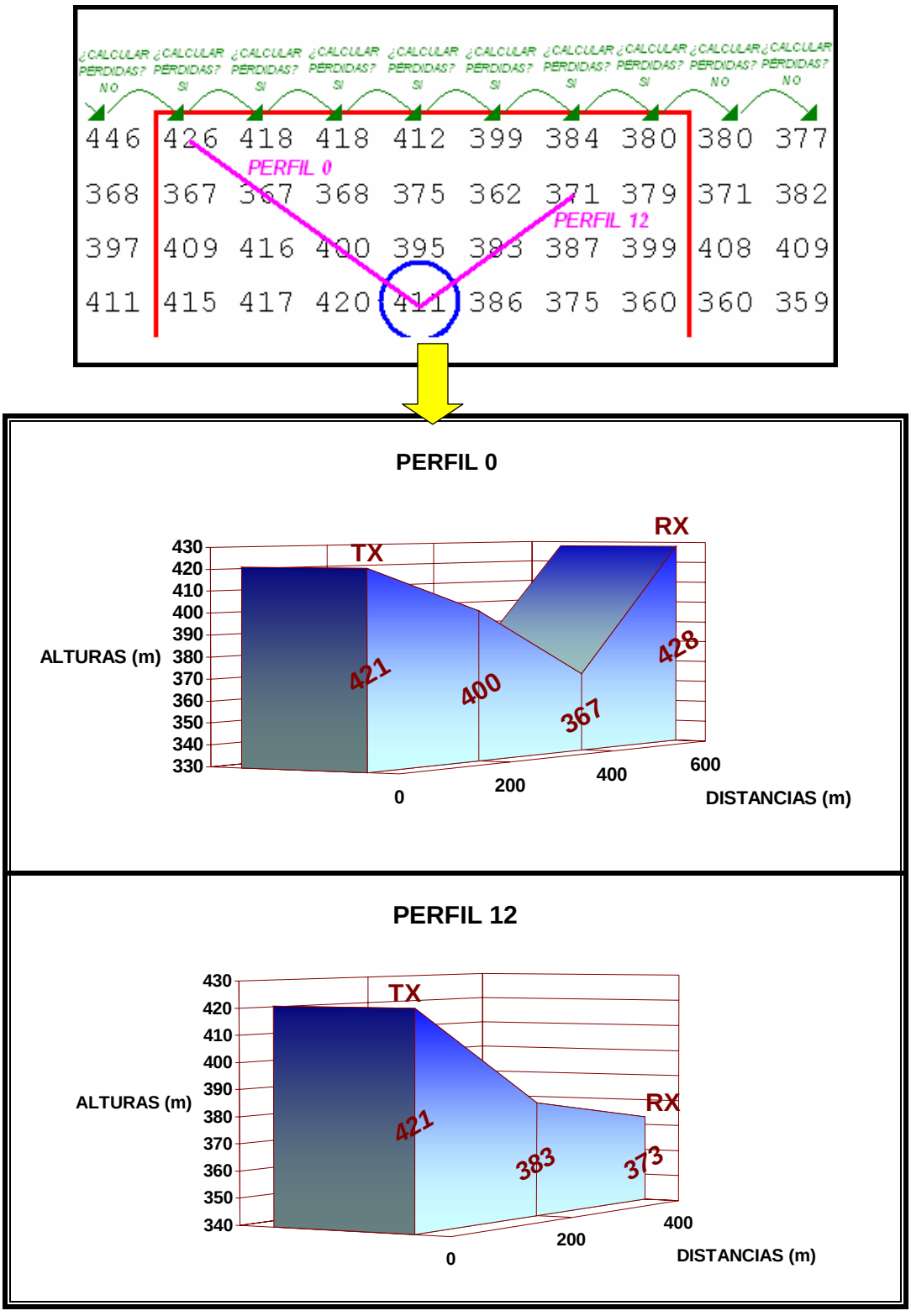


Figura 1.4. Ejemplo explicativo 1. Funcionamiento del programa original en la fila 1 del MDT (levantamiento del perfil 0 y 12).

Como nota aclaratoria, decir que en cada perfil levantado, se suman las alturas de la antena transmisora y receptora a la cota del terreno (por ejemplo, 10 y 2 metros respectivamente en estos casos).

El objetivo es cambiar este procedimiento. Se recorrerán todas las celdas de la matriz del MDT y los perfiles se levantarán de forma similar que anteriormente, pero con otra filosofía de elección:

1. Únicamente se levantará el perfil, en caso de que la celda forme parte del borde del área de cobertura a calcular, por lo que, para el ejemplo tratado, se crearán 22 perfiles (y no 41 como anteriormente).

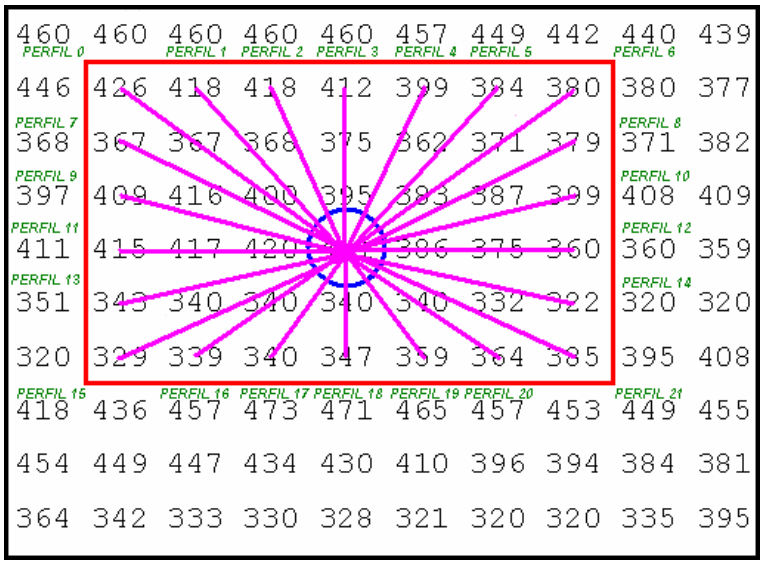


Figura 1.5. Ejemplo explicativo 1. Perfiles a levantar durante la ejecución del programa modificado.

2. Una vez se disponga del perfil, se recorrerá cada una de las celdas que lo componen, preguntando si se desean calcular las pérdidas en dicha celda; la respuesta puede ser negativa en caso que las pérdidas ya se hayan calculado (esto ocurre porque una misma celda, según su posición en la matriz, puede formar parte de varios perfiles), o afirmativa, por lo que se llamará al método que realiza dichos cálculos.

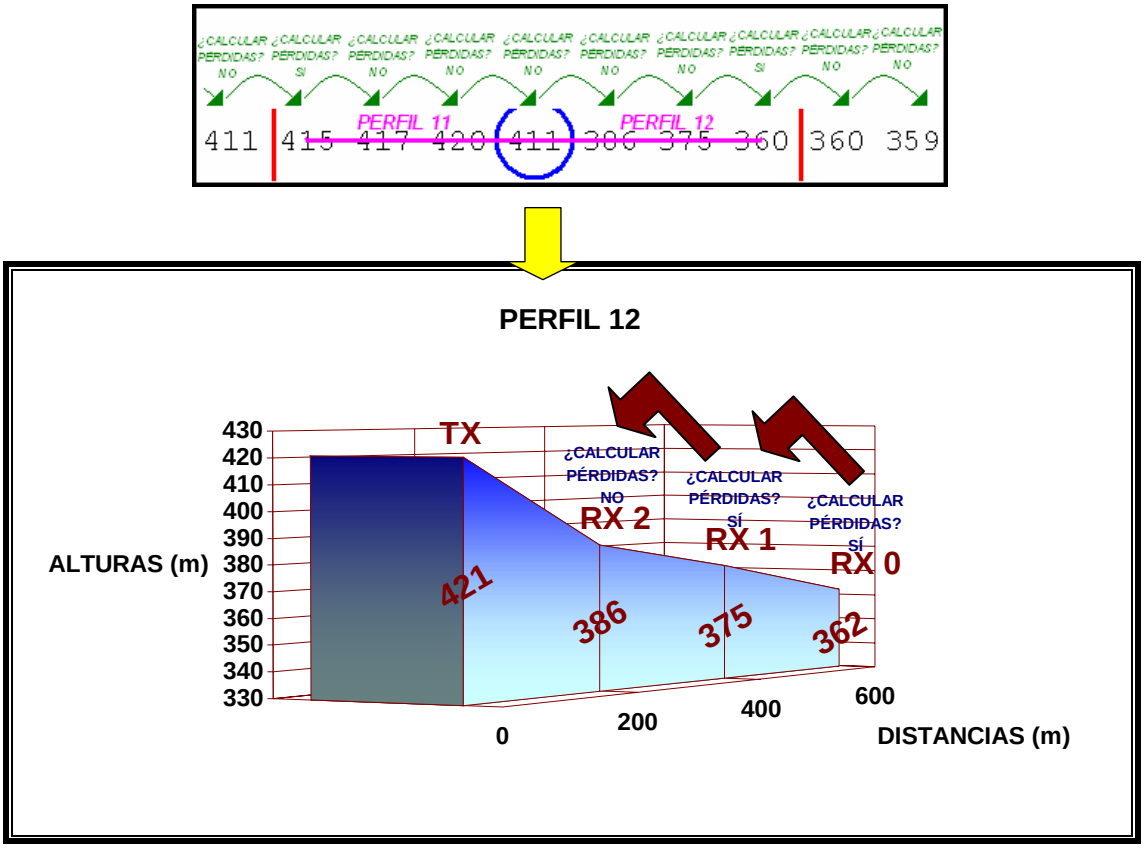


Figura 1.6. Ejemplo explicativo 1. Funcionamiento del programa modificado en la fila 4 del MDT y en "su" perfil 12 levantado.

Para este ejemplo, se calcularían las pérdidas para el RX 0 y el RX 1, pero no para el RX 2, ya que esta celda de 368 metros seguramente pertenecerá al perfil 10, el cual estará formado mediante interpolación por las alturas 421 (411+10), 386, 387 y 397 (399+2).

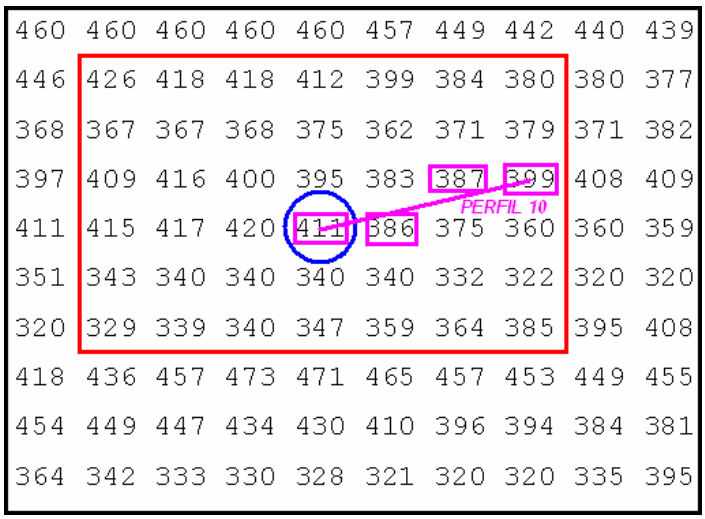


Figura 1.7. Ejemplo explicativo 1. Celdas que forman el perfil 10 del programa modificado.

Concluyendo, **levantando el perfil en un menor número de ocasiones, se ganará en velocidad de cálculo de coberturas** (en los apartados 1.4. y 2.4. se explicará todo más detalladamente).

Modificado el fichero rural, se procederá a modificar el urbano siguiendo la misma filosofía.

- *FASE CUARTA:*

Cuando se dispone del código modificado y su funcionamiento es correcto, se procede a realizar distintas simulaciones de coberturas. Se analizan tanto el archivo rural como el urbano. Las medidas se toman para áreas rectangulares y circulares, con diferentes grados de resolución del MDT:

	MDT	
	RURAL	URBANO
RESOLUCIÓN EN METROS	200	6
	100	4
	30	2

Tabla 1.1. Resoluciones en metros de los diferentes MDT utilizados en las simulaciones.

Las medidas se dividen en dos partes diferenciadas según sea el objetivo de análisis que se persiga:

1. Una primera donde el objetivo es analizar la **ganancia de velocidad de cálculo** adquirida por los ficheros como consecuencia de la optimización. Para ello, se toman los tiempos empleados, por la versión original y la optimizada, para el cálculo de diferentes tamaños de coberturas. Una vez que se dispone de la tabla con los resultados, se obtienen una serie de gráficas en las que se muestran:

- tiempos de cálculo de cada uno de los programas,
- mejora de la velocidad de programa optimizado en segundos,
- y mejora de la velocidad de programa optimizado en tanto por ciento.

2. Otra segunda en la que se obtiene el **grado de error** cometido en el cálculo de las diferentes coberturas, como consecuencia de la ganancia de velocidad. Se muestra el resultado obtenido en la interfaz gráfica de *RADIOGIS* de las coberturas calculadas por el archivo original y el optimizado. Posteriormente se presenta (también sobre la interfaz gráfica) una resta entre los diferentes valores de pérdidas obtenidos por ambos, para poder apreciar visualmente el grado de error cometido; además, irá acompañada por un recuadro donde se aprecian los valores en dB´s del error mínimo, máximo, medio, y desviación típica de dicho error.

Por último se realiza un histograma dónde se refleja en cuántas celdas del área de cobertura calculada se ha producido un error de "x" dB´s, siendo "x" diferentes valores (en dB´s) comprendidos entre el error mínimo y máximo, ambos incluidos.

- *FASE QUINTA:*

Llegada a esta fase del proyecto, se está en disposición de sacar conclusiones en cuanto a la optimización realizada, para determinar en qué medida puede ser recomendable perder precisión para ganar en velocidad de cálculo de coberturas.

En este nivel del proyecto se realiza la memoria del proyecto, intentando dotar a ésta del mayor orden y claridad posible, para que el trabajo realizado pueda ser entendido por aquellas personas que se dispongan a leerlo. Para ello, los modelos urbano y rural son tratados en apartados diferentes.

- FASE SEXTA:

Exposición y defensa del presente proyecto fin de carrera.

1.3. PALABRAS CLAVE.

RADIOGIS.

Optimización.

Velocidad.

Cobertura.

Radioenlace.

Pérdidas.

Perfil.

Rural.

Urbano.

UHF.

Otras palabras específicas: MDT, grid, método/modelo, UITR-526, COST-231, Xia/Bertoni, ArcGis.

SISTEMAS DE RADIOCOMUNICACIONES EN UHF

2. SISTEMAS DE RADIOCOMUNICACIONES EN UHF.

UHF (siglas del inglés: Ultra High Frequency, frecuencia ultra alta) es una banda del espectro electromagnético que ocupa el rango de frecuencias de 300MHz a 3GHz. *[6]*

Los diferente sistemas que funcionan en UHF son los siguientes:

1. Televisión.

Uno de los servicios UHF más conocidos por el público son los canales de televisión, tanto locales como nacionales. Según los países, algunos canales ocupan las frecuencias entre algo menos de 500MHz y unos 800 o 900MHz. *[6]*

2. Radios para uso no-profesional.

En Estados Unidos y otros países americanos, existe el servicio FRS, que permite a particulares utilizar transmisores portátiles de baja potencia para uso no-profesional. Sus equivalentes en Europa son los radiotransmisores de uso personal PMR446. Los radioaficionados también cuentan con dos bandas UHF: *[6]*

A. La banda de entre los 430 y 440MHz, y con carácter secundario; es decir, deben compartir las frecuencias con otros servicios y no son prioritarios. Esos otros servicios pueden ser por ejemplo transmisores de baja potencia para apertura de garages, repetidores de televisión para hogares y dispositivos de comunicación de baja potencia. *[6]*

B. La banda de 1200MHz *[6]*

3. Telefonía móvil.

A. Para uso público.

Históricamente, las primeras frecuencias UHF utilizadas en telefonía móvil en Europa lo fueron alrededor de los 500MHz (sistema Radiocom 2000 en Francia, sistema NMT en Escandinavia). *[6]*

Con la llegada de la norma internacional GSM, las frecuencias afectadas en UHF se sitúan alrededor de los 900MHz. *[6]*

La norma DCS1800 de telefonía móvil es similar a la GSM, sólo que la frecuencia es doble (1800MHz). Por esa misma razón, el alcance es algo inferior pero también existe más espectro para los clientes, y la denegación de conexión por falta de canales en zonas altamente pobladas es menos frecuentes. *[6]*

B. Para uso privado.

Redes privadas de la policía, bomberos, taxis, hospitales,est. Estas redes se basan en el sistema de radio trunking, en el cual las conversaciones van precedidas de un código de llamada similar a una telefónica, donde si el equipo la recibe y no es el destinatario, la emite de nuevo actuando como repetidor, y si por el contrario sí es el destinatario, se establece un circuito entre TX y Rx para asegurar la comunicación. Dependiendo del servicio instalado se puede implementar conexión a la red de telefonía pública.

El sistema trunking puede ser analógico o digital (TETRA, TETRAPOL).

ANÁLISIS DE TIEMPOS DE COMPUTACIÓN DEL MODELO RURAL

3. ANÁLISIS DE TIEMPOS DE COMPUTACIÓN DEL MODELO RURAL.

3.1. PROPAGACIÓN EN ENTORNOS RURALES.

3.1.1. Introducción.

El método de predicción de atenuación de propagación con el que se va a trabajar, requiere el conocimiento del perfil orográfico entre el transmisor y el receptor, y resulta idóneo para radioenlaces punto a punto. *[3]*

Cuando se trata de radiocomunicaciones zonales, de punto a zona, existe, en general, una gran variabilidad de los trayectos de propagación. El estudio suele efectuarse analizando perfiles a lo largo de radiales trazados desde el transmisor en distintas direcciones acimutales. Es habitual trabajar como mínimo con 12 radiales. *[3]*

3.1.2. Modelo UITR-526.

La difracción es el fenómeno que ocurre cuando una onda electromagnética incide sobre un obstáculo. La tierra y sus irregularidades pueden impedir la visibilidad entre las antenas transmisora y receptora en ciertas ocasiones. La zona oculta a la antena transmisora se denomina zona de difracción. En esta zona los campos no son nulos debido a la difracción causada por el obstáculo, y por tanto es posible la recepción, si bien con atenuaciones superiores al espacio libre. *[4]*

Para entender el concepto de difracción hay que conocer qué se entiende como zonas de Fresnel. Se definen a éstas como la región definida por los puntos del espacio que cumplen: *[4]*

$$(r_1 + r_2) - R = n * \frac{\lambda}{2} \qquad (3.1)$$

Las zonas de Fresnel son elipsoides de revolución cuyo eje mayor tiene una longitud de $R + n*(\lambda/2)$ (3.2). La intersección de las zonas de Fresnel con el plano P son circunferencias cuyo radio puede calcularse para el caso que dicho radio sea mucho menor que las distancias d_1 y d_2 como: *[4]*

$$Rf = \sqrt{n * \lambda * \frac{d_1 * d_2}{d_1 + d_2}} \qquad (3.3)$$

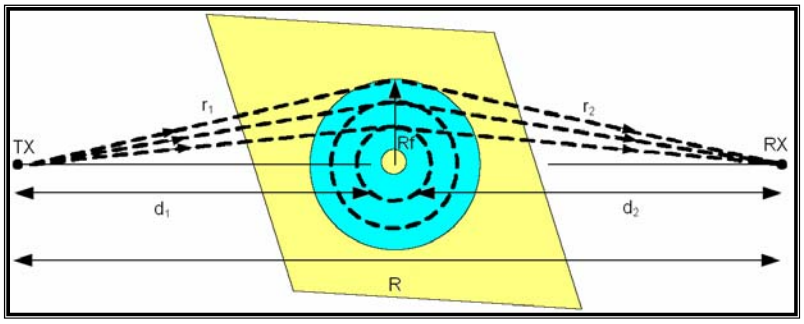


Figura 3.1. Zonas de Fresnel. *[9]*

Las fuentes equivalentes de la primera zona de Fresnel se sumarán en la antena receptora con una fase inferior de 180º. Además, son las más importantes debido a la directividad asociada a la antena. Las contribuciones de las fuentes situadas en las zonas sucesivas (n = 2, 3, 4, …) tienden a cancelarse mutuamente a pares (la 3 con la 4, la 4 con la 5, etc.). Por tanto, el radio de la primera zona de Fresnel permite definir la condición de visibilidad entre antenas, de forma que mientras no exista un obstáculo dentro de la misma se considera que la trayectoria no ha sido obstruida. Por el contrario, cuando el obstáculo se encuentra dentro de la primera zona de Fresnel existirá una disminución apreciable en la potencia recibida, por lo que se considera que la trayectoria ha sido obstruida y deberá considerarse el efecto de difracción. *[4]*

En la práctica, al estar la energía concentrada cerca del rayo directo, si el obstáculo no penetra en más de un 40% del radio de la primera zona de Fresnel se suele considerar que dicho obstáculo no contribuye significativamente a la atenuación por difracción. *[4]*

Para el estudio del cálculo de las pérdidas producidas por la difracción se empleará el método que propone la recomendación 526 de la UIT-R, la cual es válida para cualquier tipo de obstáculo. *[2]*

Para la valoración, en primera aproximación, de las pérdidas por difracción en obstáculos, se idealiza la forma de éstos, asimilándolos a una arista de espesor despreciable (arista aguda o "filo de cuchillo") o a una arista gruesa y redondeada, con un cierto radio de curvatura en la cima. *[2]*

Antes de calcular las pérdidas producidas por múltiple obstáculos, se calcularán las pérdidas producidas por un obstáculo aislado. El obstáculo conocido como "filo de cuchillo" bloquea una parte de la energía radioeléctrica, el resto será objeto de difracción entorno a la arista. *[2]*

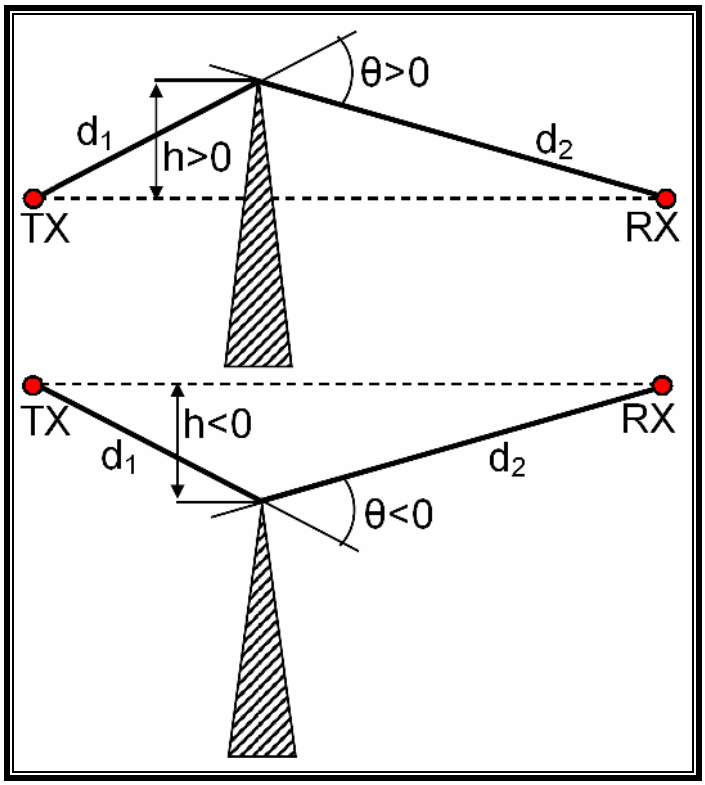


Figura 3.2. Obstrucción radioeléctrica sobre obstáculo tipo "filo de cuchillo". *[8]*

La UIT-R proporciona la forma de calcular la atenuación producida por el obstáculo en función del parámetro adimensional *v*: *[2]*

$$\upsilon = h * \sqrt{\frac{2*(d_1+d_2)}{\lambda * d_1 * d_2}}$$

h: despejamiento
λ: longitud de onda
d_1: distancia del obstáculo al emisor
d_2: distancia del obstáculo al receptor (3.4)

Se puede comprobar que v es igual a la raíz de dos veces el despejamiento normalizado $h/R1$, donde $R1$ es el radio de la primera zona de Fresnel. Expresando v en unidades usuales, resulta: *[2]*

$$\upsilon = 2{,}58 * 10^{-3} * \sqrt{\frac{f(MHz) * d(km)}{d_1(km) * d_2(km)}} * h(m) \qquad (3.5)$$

La atenuación por difracción en función de v es: *[2]*

$$L_D(\nu) = -10 * \log_{10}(0{,}5 * ((0{,}5.C(\nu))^2 + (0{,}5 - S(\nu))^2))dB \qquad (3.6)$$

$C(v)$ y $S(v)$ son las integrales de Fresnel de argumento v: *[2]*

- $C(\nu) = \int_0^{\nu} \cos\left(\frac{\pi * t^2}{2}\right) dt$ (3.7)
- $S(\nu) = \int_0^{\nu} sen\left(\frac{\pi * t^2}{2}\right) dt$ (3.8)

Afortunadamente, para las aplicaciones más usuales a la radiocomunicación se puede aproximar $L_D(v)$ por la siguiente expresión numérica (sólo válida para valores de $v > -1$): *[2]*

$$L_D(\nu) = 6{,}9 + 20 * \log_{10}(\sqrt{(\nu - 0{,}1)^2 + 1} + \nu - 0{,}1)dB \qquad (3.9)$$

En cuanto a la situación de obstáculo redondeado, cuya geometría se representa en la figura siguiente, intervienen los siguientes parámetros: *[2]*

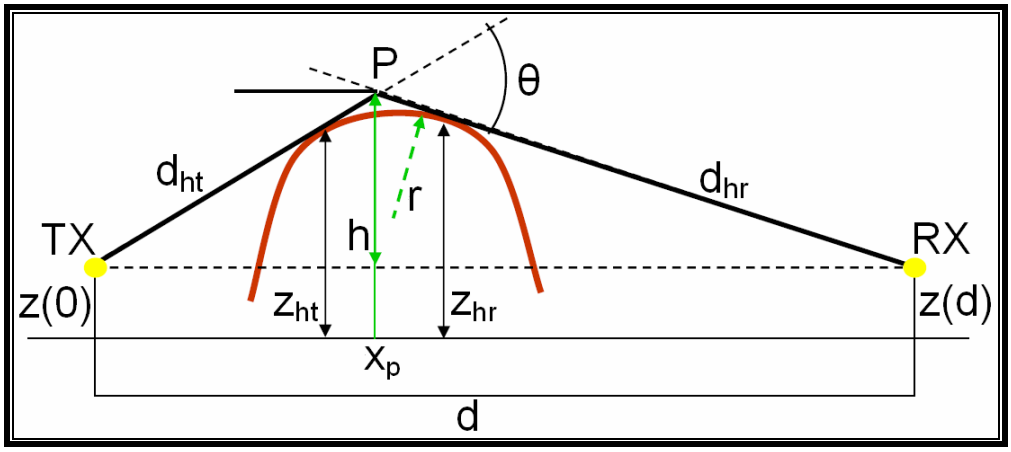


Figura 3.3. Obstrucción radioeléctrica sobre obstáculo tipo redondeado. *[8]*

- d: longitud del enlace (km).
- d_{ht} y d_{hr}: distancias del transmisor y receptor a sus horizontes respectivos (m).
- z_{ht} y z_{hr}: alturas de los puntos de horizonte (m).
- θ: ángulo de difracción (mrad).
- r: radio de curvatura del obstáculo (km).
- h: altura del punto P de intersección de las visuales trazadas desde TX y RX a sus respectivos horizontes con respecto a la línea TX-RX. *[2]*

Para la aplicación de las fórmulas de difracción sobre obstáculo redondeado hay que evaluar r y h. Se suponen conocidas las alturas de las antenas transmisora y receptora sobre el nivel del mar, $z(0)$ y $z(d)$. Además $R = K * R_0$ (3.10) (K es el factor de corrección del radio terrestre). *[2]*

El radio de curvatura del obstáculo se puede estimar mediante la expresión: *[2]*

$$r(km) = \frac{d - d_{ht} - d_{hr}}{\theta} * 10^3 \qquad (3.11)$$

donde $\theta(mrad) = \frac{z_{ht} - z(0)}{d_{ht}} + \frac{z_{hr} - z(d)}{d_{hr}} + \frac{d}{2 * R_0} * 10^3$. (3.12)

La altura *h* puede ser calculada como: *[2]*

$$h = x_p * \left(\frac{z_{ht} - z(0)}{d_{ht}} - \frac{z(d) - z(0)}{d} \right) \qquad (3.13)$$

siendo x_p la abcisa del punto *P*, dada por $x_p = \frac{d * \beta}{\theta}$. (3.14)

El ángulo β es: $\beta(mrad) = \frac{z(d) - z(0)}{d} - \frac{z(d) - z_{hr}}{d_{hr}}$. (3.15)

Falta definir los parámetros radioeléctricos siguientes para el cálculo de la difracción: *[2]*

$$m = 0{,}45708 * \frac{d_{ht} + d_{hr}}{d_{ht} * d_{hr}} * r^{2/3} * f^{-1/3} \qquad (3.16)$$

$$n = 4{,}787 * 10^{-3} * h * r^{-1/3} * f^{2/3} \qquad (3.17)$$

Es posible aproximar las distancias d_{ht} y d_{hr} por $d_1 = x_p$ y $d_2 = d - x_p$. La atenuación por difracción para el obstáculo redondeado quedará finalmente como: *[2]*

$$A(dB) = L_D(v) + T(m,n) \qquad (3.18)$$

$L_D(v)$ se evalúa mediante las expresiones ya vistas y $T(m,n)$ puede calcularse de la siguiente manera: *[2]*

$$T(m,n) = k * m^b \quad \begin{cases} k = 8{,}2 + 12 * n \\ b = 0{,}73 + 0{,}27 * (1 - e^{(-1{,}43 * n)}) \end{cases} \qquad (3.19)$$

Una vez conocidas las pérdidas producidas tanto por un obstáculo aislado en forma de filo de cuchillo, como por un obstáculo aislado redondeado, se puede pasar a analizar las pérdidas producidas por múltiples obstáculos. *[2]*

Para múltiples obstáculos el método a seguir es el siguiente propuesto por la recomendación UIT R PN 526 ampliando métodos diseñados para dos obstáculos. *[2]*

Asúmase un perfil como el de la siguiente figura: *[2]*

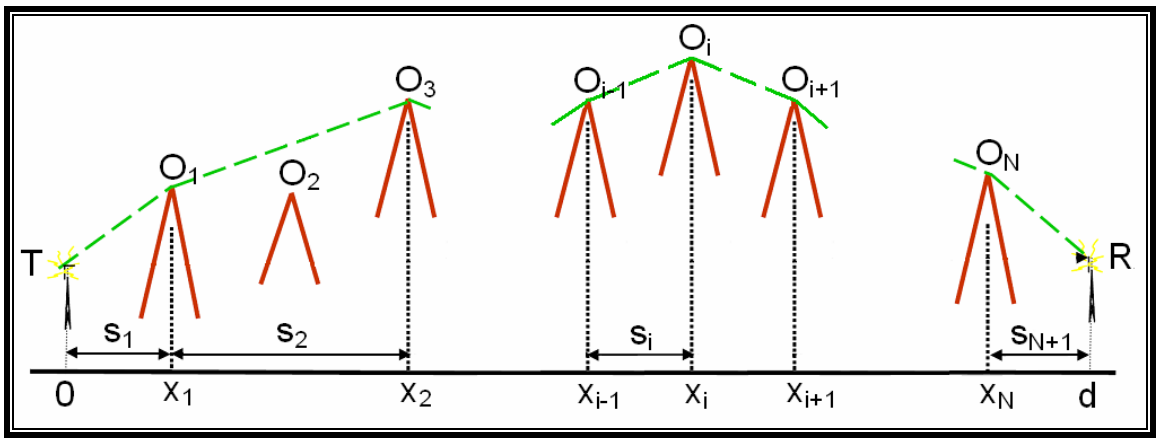


Figura 3.4. Perfil de múltiples obstáculos. *[8]*

- N: número de obstáculos.
- h_i: altura del obstáculo O_i respecto a la recta que une O_{i-1} con O_{i+1}.
- x_i: abscisa del obstáculo Oi.
- $s_i = x_i - x_{i-1}$: distancia entre los obstáculos O_{i-1} y O_{i+1}.
- $s_1 = x_1$: distancia entre el primer obstáculo y el transmisor.
- $s_{n+1} = d - x_n$: distancia del último obstáculo al transmisor (d es la distancia entre emisor y receptor). *[2]*

Para el cálculo de la atenuación, se determina el llamado "polígono funicular", que es una línea poligonal cuyos vértices son T, R y los obstáculos dominantes. *[2]*

La atenuación por difracción viene dada, para múltiples obstáculos, por: *[2]*

$$L_D(dB) = \sum_{i=1}^{n} L_D(\upsilon_i) + \sum_{i=1}^{j} L_{SD}(\upsilon`_i) - 10 * \log_{10}(C_n) \qquad (3.20)$$

siendo:

$$\upsilon_i = 2{,}58 * 10^{-3} * h_i * \sqrt{f * \frac{s_i + s_{i+1}}{s_i * s_{i+1}}} \qquad (3.21)$$

L_{sd} son las pérdidas debidas a obstáculos en el subvano O_{i-1}, O_i, O_{i+1} que queden por debajo de la línea O_{i-1}, O_{i+1} con despejamiento insuficiente. Únicamente se tendrá en cuenta un único obstáculo, el que tenga el parámetro v menos negativo (el peor caso). C_m es un factor de corrección que viene dado por: *[2]*

$$Cn = \left[\frac{s_2 * \ldots * s_N * (s_1 + s_2 + \ldots + s_{N+1})}{(s_1 + s_2) * (s_2 + s_3) * \ldots * (s_N + s_{N+1})} \right] \qquad (3.22)$$

Además, a todas estas pérdidas producidas por difracción, se suman las **pérdidas debidas al espacio libre**. *[2]*

3.1.3. Modelo espacio libre.

Utilizando este método de propagación, las pérdidas se calcularán sin tener en cuenta ningún obstáculo suponiendo un medio dieléctrico homogéneo, teniendo en cuenta la siguiente fórmula: *[4]*

$$L_{bf} = 32{,}45 + 20 * \log_{10} f\,(\text{MHz}) + 20 * \log_{10} d\,(\text{km}) \qquad (3.23)$$

El método de espacio libre se puede utilizar para cualquier rango de frecuencias y alturas del transmisor y receptor. *[4]*

3.2. CÓDIGO COMPLETO DEL MODELO RURAL ORIGINAL.

En este apartado de presenta el código original del proyecto UITR-526, sobre el cual se ha trabajado. Se puede observar incluso los comentarios en referencia a un gran número de líneas de código.

En un apartado posterior (punto 3.3) se expondrá el mismo, pero con las modificaciones adecuadas para conseguir el aumento de velocidad en el cálculo de pérdidas. De esta forma podrán apreciarse las diferencias, aunque aún más adelante (apartado 3.4), éstas serán explicadas con mayor detalle.

3.2.1. Fichero principal.cpp.

```
#include <time.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <locale.h>


#define PI abs(acos(-1.0))
#define Re 8.5e6 /*Radio efectivo de la Tierra (m)*/

struct lconv *leng;
char sep,no_sep;


/***FUNCIONES DE LECTURA Y ESCRITURA***/

void lee_float (FILE*ent,float *pf){
   char s[40],*p;
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%f",pf);
   }

void lee_double (FILE*ent,float *pf){
   char s[40], *p;
```

```
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%lf",pf);
   }

void escribe_float (FILE*sal,float *pf){
   char s[40],*p;
   sprintf(s,"%.0f",*pf);
   while (p=strchr(s,no_sep))  *p=sep;
   fprintf(sal,"%s",s);
   }


/*****************MAIN*************/

main (int argc,char *argv[]){

float CalculaAtenuacion(float *perfil, float cellsize, float frec, float K, float
*perfil_flecha, float *vector_distancias, int npuntos, float R, int *indices_ptoscorte,
float Rt);


/***FICHEROS UTILIZADOS***/

FILE *grid;
FILE *fdatos;
FILE *resultados;
FILE *comprueba;
FILE *mdt;
FILE *tiempo;


/***VARIABLES AUXILIARES***/

float pi = 3.141592654;
int i,j;
int fr,cr; /*Fila y columna del rx*/
long int prod,ind,prod1,ind1,indtx;
float *mapa,*campo;
char *tira;
char diag[160];
float azimut, elevacion;

float G=0;
float temp;
int dx,dy,e1;
float e;
int *x,*y;
float paso,aux;
```

```
float *perfil;
float *perfil_flecha;
float *vector_distancias;
int *indices_ptoscorte;
float atenuacion;
float paso2;
int obstaculo; /*Se incluye en el fichero de datos que se le pasa*/
float k;
clock_t start, end;


/***PARAMETROS OBTENIDOS DEL FICHERO DE DATOS***/

float frecuencia; /*MHz*/
float ht,hr; /*Alturas del tx y rx*/
int nfils,ncols; /*Número de filas y columnas del grid*/
int ft,ct; /*Fila y columna donde se encuentra el tx*/
float celda;/*Lado de la celda en metros*/
float R; /*Distancia tx-rx, en el suelo (en m)*/
float Rtotal; /*Distancia tx-rx (en m), en diagonal*/
int ncols2,nrows2,nodata2;
double xllcorner2,yllcorner2,cellsize2; /*Se guardan los valores del MDT genérico*/
int ncols1,nrows1,nodata1;
double xllcorner1,yllcorner1,cellsize1; /*Se guarda la cabecera del fichero de alturas*/
int x1,y1;
int filasab,filasar,coli,cold;
int fa,fab,ci,cd;
float radio;
float figura,xllcorner,yllcorner;

start=clock();
leng=localeconv(); /*Pide el lenguaje actual*/
sep=*(leng->decimal_point);
if (sep==',')no_sep='.';
else no_sep=',';


/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO DE DATOS***/

frecuencia=atof(argv[5]);
ht=atof(argv[6]);
hr=atof(argv[7]);
nfils=atoi(argv[8]);
ncols=atoi(argv[9]);
ct=atoi(argv[10]);
ft=atoi(argv[11]);
celda=atof(argv[12]);
obstaculo=atof(argv[13]);
k=atof(argv[14]);
```

```
azimut=atof(argv[15]);
elevacion=atof(argv[16]);

figura=atof(argv[18]);
radio=atof(argv[19]);
xllcorner=atof(argv[20]);
yllcorner=atof(argv[21]);


/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO MDT PARA COPIARLOS EN EL FICHERO
RESULTADOS***/

if ((mdt=fopen(argv[4],"r"))!=NULL){ /*Abre el fichero de campo para escritura*/

   if ((resultados=fopen(argv[3],"w"))!=NULL){

      tira=(char*)calloc(80,sizeof(char));

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&ncols2);
      fprintf(resultados,"%d\n",ncols2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&nrows2);
      fprintf(resultados,"%d\n",nrows2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&xllcorner2);
      fprintf(resultados,"%lf\n",xllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&yllcorner2);
      fprintf(resultados,"%lf\n",yllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s      ",tira);
      lee_double(mdt,&cellsize2);
      fprintf(resultados,"%lf\n",cellsize2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s  ",tira);
      fscanf(mdt,"%d\n",&nodata2);
      fprintf(resultados,"%d\n",nodata2);

      fclose(resultados);
```

```
      free(tira); /*Libera la memoria que reservamos con calloc*/
   }

   else{
      printf("No se puede abrir fichero de salida\n");
      exit(1);
   }

   fclose(mdt);
}


/***LECTURA DE LA CABECERA DEL FICHERO ALTURAS***/

if ((grid=fopen(argv[2],"r"))!=NULL){

   tira=(char*)calloc(80,sizeof(char));

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&ncols1);

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nrows1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&xllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&yllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&cellsize1);

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nodata1);

   free(tira); /*Libera la memoria que se ha reservado con calloc*/


   /***RESERVA DE MEMORIA PARA EL MDT (MAPA) Y EL MAPA DE CAMPO (CAMPO) RESULTADO***/

   prod=(long) nrows1*ncols1;
   mapa=(float*)calloc(prod,sizeof(float));
   campo=(float*)calloc(prod,sizeof(float));
   indtx=(long)ncols1*ft+ct;

   if (!mapa||!campo){
      printf("No queda memoria disponible para ciudad o campo\n");
      exit(1);
```

```
   }

   for (i=0;i<nrows1;i++)
   for (j=0;j<ncols1;j++){
      ind=(long) ncols1*i+j; /*Implementa una matriz con un vector*/
      lee_float(grid,&temp);
      if(fabs(temp+9999)<1)temp=0;

      if (ind==indtx) mapa[ind]=temp+ht;
      else {
         mapa[ind]=temp; /*Almacena en mapa la información del MDT*/
      }

   }

   fclose(grid);
}

else{
   printf("No se puede abrir fichero de entrada\n");
   exit(1);
}

fa=((yllcorner1+nrows1*cellsize1)-(yllcorner+nfils*cellsize1))/cellsize1;
fab=nrows1-fa-nfils;
ci=(xllcorner-xllcorner1)/cellsize1;
cd=ncols1-ci-ncols;


/***BUCLE PRINCIPAL***/

for (fr=0;fr<nrows1;fr++)
for (cr=0;cr<ncols1;cr++){
   ind=(long)ncols1*fr+cr;

   if(ind!=indtx) mapa[ind]=mapa[ind]+hr;

   if ((mapa[ind]>=hr) && (fr>fa-1) && (fr<fa+nfils) && (cr>ci-1) &&
   (cr<ci+ncols)){ /*Sólo se calcula en la zona seleccionada*/


      /***CÁLCULO DE LA DISTANCIA TX-RX***/

      dx=cr-ct;
      dy=fr-ft;
      e=(float)sqrt(dx*dx+dy*dy); /*Distancia tx-rx (en numero de celdas)*/
      R=e*(celda);
      Rtotal=(float)sqrt(R*R+(mapa[indtx]-mapa[ind])*(mapa[indtx]-mapa[ind]));
      if ((figura==1) && (R > radio)) campo[ind]=-9999 ;
```

```
else{


   /***RESERVA DE MEMORIA***/

   e1=floor(e)+1; /*Tamaño de los vectores x e y*/
   x=(int*)calloc(e1,sizeof(int)); /*En x[] se guardan las columnas y en y[] las
                                    filas del gris que determinan los puntos
                                    pertenecientes al perfil*/
   y=(int*)calloc(e1,sizeof(int));
   if(!x||!y){
      printf("No queda memoria disponible para coordenadas\n");
      exit(1);
   }


   /***CREACIÓN DEL VECTOR X***/

   paso=(e==0) ? 0 : dx/floor(e);
   for (i=0;i<(e1-1);i++){
      aux=ct+i*paso-floor(ct+i*paso);
      if (aux<0.5) x[i]=floor(ct+i*paso);
      else x[i]=ceil(ct+i*paso);
   }

   x[e1-1]=cr; /*El último punto no se interpola, al igual que el primero*/


   /***CREACIÓN DEL VECTOR Y***/

   paso=(e==0) ? 0 : dy/floor(e);
   for (i=0;i<(e1-1);i++){
      aux=ft+i*paso-floor(ft+i*paso);
      if (aux<0.5) y[i]=floor(ft+i*paso);
      else y[i]=ceil(ft+i*paso);
   }

   y[e1-1]=fr; /*Los elementos de los vectores x e y están ordenados de menor a
                mayor*/


   /***SE HALLA EL PERFIL***/

   perfil=(float*)calloc(e1,sizeof(float));
   perfil_flecha=(float*)calloc(e1,sizeof(float));
   vector_distancias=(float*)calloc(e1,sizeof(float));
   indices_ptoscorte=(int*)calloc(e1,sizeof(float));

   if (!perfil){
```

```
            printf("No queda memoria disponible para perfil\n");
            exit(1);
         }

         for(j=0;j<e1;j++){
            ind=(long) ncols1*y[j]+x[j];
            perfil[j]=mapa[ind];
         }


         /***CREACIÓN DEL VECTOR DE DISTANCIAS***/

         paso2=(e==0) ? 0 : R/floor(e);
         for (i = 0; i < (e1-1); i++) {
            vector_distancias[i]=i*paso2;
         }

         vector_distancias[e1-1]=R;  /*Como anteriormente, el último punto no se
                                       interpola,al igual que el primero*/

         free(x);
         free(y);

         G=0;

         if (e1==1) {
            atenuacion=-9999;
            campo[ind]=atenuacion;
         }

         else{

            atenuacion=CalculaAtenuacion(perfil,celda,frecuencia,k,perfil_flecha,
            vector_distancias,e1,R,indices_ptoscorte,Rtotal);
            campo[ind]=atenuacion - G;
         }

         free (perfil);
         free (perfil_flecha);
         free (vector_distancias);
         free (indices_ptoscorte);
      }

      if(ind!=indtx)  mapa[ind]=mapa[ind]-hr;  /*Se quita la altura del rx para la
                                                 siguiente iteración*/
   }

   else campo[ind]=-9999;
} /***FIN DEL BUCLE PRINCIPAL***/
```

```
/***ESCRITURA DEL FICHERO DE CAMPO EN DISCO***/

if ((resultados=fopen(argv[3],"a"))!=NULL){

   filasar=((yllcorner2+cellsize2*nrows2)-(yllcorner1+cellsize1*nrows1))/cellsize2;
   filasab=nrows2-nrows1-filasar;
   coli=(xllcorner1-xllcorner2)/cellsize2;
   cold=ncols2-ncols1-coli;

   for (x1=0;x1<filasar;x1++){
      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   for (i=0;i<nrows1;i++){
      for (y1=0;y1<coli;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      for (j=0;j<ncols1;j++){
         ind1=(long) ncols1*i+j;
         escribe_float(resultados,&(campo[ind1]));
         fprintf(resultados," ");
      }

      for (y1=0;y1<cold;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   for (x1=0;x1<filasab;x1++){
      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }
   fclose (resultados);
   printf("El campo está  en disco\n");
}
```

```
else{
   printf("No se puede abrir fichero de salida\n");
   exit (1);
}

free(campo);
free(mapa);

comprueba=fopen(argv[22],"w");
fprintf(comprueba,"\n");

fclose (comprueba);
end = clock();

if  ((tiempo=fopen("c:\\arcgis\\RAGIS\\tiempo.txt","w"))!=NULL){  /*Abre  el  fichero  de
                                                                   campo para escritura*/
   fprintf(tiempo,"The time was: %f\n", (end - start) / CLK_TCK);
   fclose(tiempo);
}

} /***FIN DE MAIN***/
```

3.2.2. Fichero calculaatenuacion.cpp.

```
#include <dos.h>
#include <windows.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define pi 3.141592654f


/***1.- MÉTODO CALCULAPERFIL_FLECHA***/

int  CalculaPerfil_flecha(float  *perfil,  int  e1,  float  K,  float  cellsize,float
*perfil_flecha) {

   const float RadioTierra = 6370000.0;
   float flecha;
   float F1;
   int i;

   F1 = cellsize * cellsize / (2 * K * RadioTierra);
```

```
   for (i = 0; i < e1; i++) {
      flecha = (i * (e1 - i - 1)) * F1;
      perfil_flecha[i] =perfil[i] + flecha;
   }

   return 0;
}


/***2.- MÉTODO LOS***/

int LOS(float *dis, float *per, int longitud){

   float tx_X, tx_Y, rx_X, rx_Y;
   float *recta1;
   int aux = 0;
   int vision, r;
   double p;
   recta1=(float*)calloc(longitud,sizeof(float));

   //SE CALCULAN LAS COORDENADAS DEL TX
   tx_X=dis[0];
   tx_Y=per[0];

   //SE CALCULAN LAS COORDENADAS DEL RX
   rx_X=dis[longitud-1];
   rx_Y=per[longitud-1];

   //SE CALCULA LA PENDIENTE DE LA RECTA
   if(tx_X!=rx_X)  p=(double)(tx_Y-rx_Y)/(tx_X-rx_X);
   else p=0;

   //SE CALCULA LA RECTA FICTICIA QUE UNE TX Y RX PARA DETERMINAR SI HAY PTOS DE CORTE
   for (r=0; r<longitud; r++){
      recta1[r]=tx_Y+p*(dis[r]-tx_X);
   }

   //SE CALCULA SI HAY PTOS DE CORTE
   for(r = 0; r<longitud; r++){
      if(recta1[r]<per[r]){ //NO HAY VISIÓN DIRECTA
         vision = 0;
         break;
      }

      else vision = 1;
   }

   free(recta1);
   return vision;
```

```
}


/***3.- MÉTODO CALCULAFUNICULO***/

int CalculaFuniculo(float *perfil_flecha, int e1, float cellsize, float
*vector_distancias, int *indices_ptoscorte) {

   int *cortes, *cd, *ci;
   int indicemax,indicemaxI,indicemaxD;
   int i=0,j=0, n=0, m=0;
   float maximoY, maximoX, paso;
   float tx_X, tx_Y, rx_X, rx_Y;
   double p_tx, p_rx;
   float *recta;
   int cont=0; /*Índice del vector 'cortes'. Indicará el tamaño del vector*/
   int auxiliar = 0;
   int s=0, k = 0; /*Índice de los vectores 'cd' y 'ci'. Indicará el tamaño del vector*/
   int a=0; /*Índice del vector 'indices_ptoscorte'. Indicará el tamaño del vector*/
   int b=0; /*Índice de los vectores 'dist' y 'perf'. Indicará el tamaño del vector*/
   int  sigue=0;  /*Variable que controlará el bucle while. Cuando no haya cortes
                  sigue=1*/
   int iI, iD;
   float Xo1,Yo1,Xo2,Yo2;
   float *dist, *perf;
   int visionD, visionI;
   double *pendientes;

   recta=(float*)calloc(e1,sizeof(float));
   cortes=(int*)calloc(e1,sizeof(float));
   cd=(int*)calloc(e1,sizeof(float));
   ci=(int*)calloc(e1,sizeof(float));
   pendientes=(double*)calloc(e1,sizeof(double));
   dist = (float*)calloc(e1,sizeof(float));
   perf = (float*)calloc(e1,sizeof(float));

   if(!recta||!cortes||!ci||!cd||!pendientes||!dist||!perf)
   printf("No queda memoria disponible\n");

   maximoY = 0;
   for (i = 0; i < e1-1; i++) {
      if(perfil_flecha[i] > maximoY){
         maximoY=perfil_flecha[i];
         indicemax = i;
      }

   } /*Ya se tiene el índice del punto más alto del perfil*/

   maximoX = vector_distancias[indicemax];
```

```
//SE CALCULAN LAS COORDENADAS DEL TX
tx_X=vector_distancias[0];
tx_Y=perfil_flecha[0];

//SE CALCULAN LAS COORDENADAS DEL TX
rx_X=vector_distancias[e1-1];
rx_Y=perfil_flecha[e1-1];

//SE CALCULA LA PENDIENTE DE LA RECTA QUE UNE EL PUNTO MÁXIMO CON EL TX
if(tx_X!=maximoX) p_tx=(double)(tx_Y-maximoY)/(tx_X-maximoX);
else p_tx=0;

//SE CALCULA LA PENDIENTE DE LA RECTA QUE UNE EL PUNTO MÁXIMO CON EL RX
if(rx_X!=maximoX)  p_rx=(double)(rx_Y-maximoY)/(rx_X-maximoX);
else p_rx=0;

//SE CREA UNA RECTA INICIAL PARA DETERMINAR LOS PUNTOS DE CORTE
for (i = 0; i<=indicemax;i++){
   recta[i]=maximoY + p_tx*(vector_distancias[i]-maximoX);
}

for (i = (indicemax + 1); i<e1; i++){
   recta[i]=maximoY + p_rx*(vector_distancias[i]-maximoX);
}

//SE DEFINE UNA VARIABLE QUE VA A IR GUARDANDO LOS ÍNDICES DE LOS PUNTOS DE CORTE QUE
//DEFINEN EL NUEVO POLÍGONO FUNICULAR
a=0;
indices_ptoscorte[a++] = 0;
if((indicemax!=0) && (indicemax!=e1-1))  indices_ptoscorte[a++] = indicemax;
indices_ptoscorte[a++] = e1-1;

indicemaxI=indicemax;
indicemaxD=indicemax;
cont=0;

//SE  CALCULAN  LOS  ÍNDICES  DE  LOS  PTOS  QUE  CORTAN  CON  LA  RECTA  INICIAL  Y  SE
//ALMACENAN EN 'CORTES'
for(i = 0; i<e1; i++){

   if(recta[i]<perfil_flecha[i]){
      cortes[cont++]=i;
      sigue=1;
   }

}

while (sigue==1){
```

```
      //SE SEPARAN LOS ÍNDICES DE LOS PUNTOS DE CORTE EN LOS QUE ESTÁN A LA DERECHA Y A
      //LA IZQUIERDA
      s=k=0;
      for(i = 0; i<cont; i++){
         if(cortes[i]<indicemaxI) ci[s++]=cortes[i];
         if(cortes[i]>indicemaxD) cd[k++]=cortes[i];
      }

////////////////////// FUNÍCULO DERECHO ////////////////////////

      if (k>0){

         for (i=k-1; i>=0 ;i--){
            b=0;
            for (m=indicemaxD; m<=cd[i]; m++){
               dist[b] = vector_distancias[m];
               perf [b++] = perfil_flecha[m];
            }

            visionD=LOS(dist,perf,b);

            if(visionD==1){
               iD=cd[i];
               indicemaxD=iD;
               break;
            }

         }

         indices_ptoscorte[a++]=iD;
      }

////////////////////// FUNÍCULO IZQUIERDO ////////////////////////

      if(s>0){

         for (i=0; i<s ;i++){
            b=0;
            for (n=ci[i]; n<= indicemaxI; n++){
               dist[b] = vector_distancias[n];
               perf[b++] = perfil_flecha[n];
            }

            visionI=LOS(dist,perf,b);

            if(visionI==1){
               iI=ci[i];
               indicemaxI=iI;
               break;
```

```
            }

         }

         indices_ptoscorte[a++]=iI;
      }


      //SE ALMACENAN LOS PTOS DE CORTE ORDENADOS EN EL VECTOR 'INDICES_PTOSCORTE' QUE
      //TAMBIÉN CONTIENE AL TX Y AL RX
      for(j = 0; j<a; j++){
         for(i = 0; i<(a-1); i++){
            if(indices_ptoscorte[i]>indices_ptoscorte[i+1]){
               auxiliar = indices_ptoscorte[i+1];
               indices_ptoscorte[i+1]=indices_ptoscorte[i];
               indices_ptoscorte[i]=auxiliar;
            }

         }

      }


      //SE CALCULAN LAS PENDIENTES ENTRE CADA DOS OBSTÁCULOS CONSECUTIVOS
      for (i = 0; i<(a-1); i++){
         Xo1=vector_distancias[indices_ptoscorte[i]];
         Yo1=perfil_flecha[indices_ptoscorte[i]];
         Xo2=vector_distancias[indices_ptoscorte[i+1]];
         Yo2=perfil_flecha[indices_ptoscorte[i+1]];
         if ((Xo2-Xo1)!=0)
         pendientes[i]=(double)(Yo2-Yo1)/(Xo2-Xo1);
      }

      //SE CALCULA DE NUEVO LA RECTA QUE DEFINE EL POLÍGONO COMPLETO (FORMADA POR RECTAS
      //CALCULADAS ENTRE DOS OBSTÁCULOS CONSECUTIVOS) Y SE COMPRUEBA SI EXISTE ALGÚN
      //PUNTO DE CORTE
      for (j = 0; j<(a-1); j++){
         for (i = indices_ptoscorte[j]; i<=indices_ptoscorte[j+1]; i++) {
            recta[i] = perfil_flecha[indices_ptoscorte[j]]+pendientes[j]*
            (vector_distancias[i]-vector_distancias[indices_ptoscorte[j]]);
         }

      }

      //SE CALCULAN DE NUEVO LOS ÍNDICES DE LOS PTOS DE CORTE
      sigue=0;
      cont = 0;
      for(i = 0; i<e1; i++){
         if(recta[i]<perfil_flecha[i]) {
```

```
            cortes[cont++]=i;
            sigue=1;
         }

      }

   } //FIN DEL WHILE

   free(perf);
   free(dist);
   free(recta);
   free(cortes);
   free(cd);
   free(ci);
   free(pendientes);

   return a;
}


/***4.- MÉTODO ESPACIO_LIBRE***/

float espacio_libre(float f, float d){
   float L;

   return L = 32.45 + 20*log10(f) + 20*log10(d);
}


/***5.- MÉTODO CALCULAATENUACIÓN***/

float CalculaAtenuacion(float *perfil, float cellsize, float frec, float K, float
*perfil_flecha, float *vector_distancias, int npuntos, float R, int *indices_ptoscorte,
float Rt) {

   float *flecha;
   float Ltotal=0, LD=0, L0=0, Llluvia=0, Lgases=0, Ldi=0, sumatorioLD=0;
   int vision;
   float roo = 7.5; /*Roo: densidad del vapor de agua en gr/m3 (7.5)*/
   float tasa = 25; /*R=25 mm/h*/
   char polarizacion = 'h';
   int i, j;
   float *alturas_obs;
   float *distancias_obs;
   int num_vanos;
   float d1, d2, vi, hi, tang, vimax;
   float sumatorioLsdi=0, Lsdi=0, Lsd_max=0;
   double Cn=0;
   double a, b, c;
```

```
   float htx, hrx, posicion_tx, posicion_rx;
   int long_ptos,l;

   flecha = (float*)calloc(npuntos,sizeof(float));
   alturas_obs = (float*)calloc(npuntos,sizeof(float));
   distancias_obs = (float*)calloc(npuntos,sizeof(float));

   if (K == (float)0){
      return -1; /*Error, la Tierra no es un punto*/
   }


   /***LLAMADA AL MÉTODO CALCULAPERFIL_FLECHA***/

   CalculaPerfil_flecha(perfil, npuntos, K, cellsize, perfil_flecha);
   for (i = 0; i < npuntos; i++)  flecha[i] = perfil_flecha[i]-perfil[i];

////////////////////////////////////ATENUACIÓN/////////////////////////////////////////

   R=R/1000; /*Distancia entre tx y rx en km*/
   Rt=Rt/1000;


   /***LLAMADA AL MÉTODO ESPACIO_LIBRE***/

   L0 = espacio_libre(frec, Rt);


/***LLAMADA AL MÉTODO LOS (devuelve un 1 si hay vision directa y un 0 si no la hay)***/

   vision = LOS(vector_distancias, perfil_flecha, npuntos);

////////////////////////////////////CASO NLOS/////////////////////////////////////////

   if (vision == 0){


      /***LLAMADA AL MÉTODO CALCULAFUNICULO***/

      long_ptos = CalculaFuniculo(perfil_flecha, npuntos, cellsize,
      vector_distancias, indices_ptoscorte);

////////////////PÉRDIDAS POR DIFRACCIÓN Y DESPEJAMIENTO INSUFICIENTE///////////////

      for(i = 0; i<npuntos; i++) vector_distancias[i]=vector_distancias[i]/1000;

      for (i = 0; i<long_ptos; i++){
         alturas_obs[i] = perfil_flecha[indices_ptoscorte[i]];
         distancias_obs[i] = vector_distancias[indices_ptoscorte[i]];
```

```
}

/*Cálculo de las pérdidas por difracción. Si no existe visión directa entre TX y
RX, habrá que calcular las pérdidas por difraccion, por despejamiento
insuficiente, y el factor Cn.para calcular las pérdidas por obstáculo múltiple*/

num_vanos=long_ptos-2; //NÚMERO DE VANOS (QUITANDO TX Y RX)
sumatorioLD=0;

//SE CALCULAN EN PRIMER LUGAR LAS PÉRDIDAS POR DIFRACCIÓN DEBIDAS A CADA VANO

for (i = 1; i<=num_vanos; i++){
   d1 = distancias_obs[i]-distancias_obs[i-1];
   d2 = distancias_obs[i+1]-distancias_obs[i];
   tang = (alturas_obs[i+1]-alturas_obs[i-1])/(d1+d2);
   hi = tang*d1+alturas_obs[i-1]-alturas_obs[i];
   hi = -hi;
   vi = 2.58*pow(10,-3)*hi*sqrt(frec*((d1+d2)/(d1*d2)));
   Ldi = 6.9+20*log10(sqrt(pow((vi-0.1),2)+1)+vi-0.1);
   sumatorioLD = sumatorioLD+Ldi;
} //FIN DEL FOR


//SE CALCULAN LAS PÉRDIDAS POR DESPEJAMIENTO INSUFICIENTE

Lsd_max=sumatorioLsdi=0;

for (i=0; i<long_ptos-1; i++){

   if(indices_ptoscorte[i]!=indices_ptoscorte[i+1]-1){
      vimax=-100000;
      l= indices_ptoscorte[i+1]-indices_ptoscorte[i]+1;
      if(l<=3) Lsdi=Lsd_max=0;
      else{
         for (j=indices_ptoscorte[i]+1; j<=indices_ptoscorte[i+1]-1; j++){

            d1=vector_distancias[j]-vector_distancias[indices_ptoscorte[i]];
            d2= vector_distancias [indices_ptoscorte[i+1]]- vector_distancias [j];

            tang=(perfil_flecha[indices_ptoscorte[i+1]]-perfil_flecha
            [indices_ptoscorte[i]])/(d1+d2);

            hi=tang*d1+perfil_flecha[indices_ptoscorte[i]]-perfil_flecha[j];
            hi=-hi;
            vi = 2.58*pow(10,-3)*hi*sqrt(frec*((d1+d2)/(d1*d2)));
            if(vi>vimax) vimax=vi;
         }

         if(vimax>-1) Lsdi=6.9+20*log10(sqrt(pow((vimax-0.1),2)+1)+vimax-0.1);
```

```
            else Lsdi =0;

            if(Lsdi>Lsd_max) Lsd_max=Lsdi;
         }

      }

   } //FIN DEL FOR

   //CÁLCULO DEL FACTOR DE CORRECCIÓN CN

   if(num_vanos>1){ //SE CALCULA EL FACTOR DE CORRECCIÓN SÓLO SI EXISTE MÁS DE UN
                    //OBSTÁCULO

      a=1;
      b=0;
      c=1;

      for (i=0; i<long_ptos; i++){
         if(i<long_ptos-3){
            a=a*(distancias_obs[i+2]-distancias_obs[i+1]);
         }

         b=distancias_obs[long_ptos-1];
         if (i<long_ptos-2){
            c=c*(distancias_obs[i+2]-distancias_obs[i]);
         }

      }

      Cn=(double)(a*b)/c;

      //PÉRDIDAS DE DIFRACCIÓN LD

      LD = sumatorioLD + Lsd_max - 10*log10(Cn);
   }

   else{
      LD = sumatorioLD + Lsd_max;
   }

   Ltotal=LD+L0;
} //FIN DEL IF

////////////////////////////////////CASO LOS/////////////////////////////////////

else if (vision == 1){

   //PÉRDIDAS POR DIFRACCIÓN
```

```
      LD = 0;

      //PÉRDIDAS POR DESPEJAMIENTO INSUFICIENTE
      for(i = 0; i<npuntos; i++) vector_distancias[i]=vector_distancias[i]/1000;

      htx=perfil_flecha[0];
      hrx=perfil_flecha[npuntos-1];
      posicion_tx= vector_distancias[0];
      posicion_rx= vector_distancias[npuntos-1];

      Lsd_max=0;

      l=npuntos;
      if(l<=3) Lsdi=Lsd_max=0;
      else{
         for (i=1; i<=npuntos-2;i++){
            d1=vector_distancias[i]-posicion_tx;
            d2=posicion_rx - vector_distancias[i];
            tang=(hrx-htx)/R;
            hi=tang*d1+htx-perfil_flecha[i];
            hi=-hi;
            vi=2.58e-3*sqrt((frec*(d1+d2))/(d1*d2))*hi;
            if(vi>-1) Lsdi=6.9+20*log10(sqrt(pow((vi-0.1),2)+1)+vi-0.1);
            else Lsdi = 0;
            if(Lsdi>Lsd_max) Lsd_max=Lsdi;
         }

      } //FIN DEL ELSE

      Ltotal=LD+L0+Lsd_max;
   } //FIN DEL ELSE IF

   free(distancias_obs);
   free(alturas_obs);
   free(flecha);

   return  Ltotal;
}
```

3.3. CÓDIGO COMPLETO DEL MODELO RURAL MODIFICADO.

A continuación se presenta el código modificado para conseguir, como ya se ha explicado anteriormente, reducir los tiempos de cálculo de pérdidas.

Tanto en el fichero "principal.cpp" como "calculaatenuacion.cpp", aparecen líneas comentadas. Estas líneas reflejan todos y cada uno de los cambios realizados con respecto al código de partida.

Aún así, en el siguiente apartado se explicará de manera exhaustiva el funcionamiento del programa, haciendo especial hincapié en las modificaciones incorporadas que diferencian al proyecto optimizado del anterior.

3.3.1. Fichero principal.cpp.

```
#include <time.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <locale.h>

#define PI abs(acos(-1.0))
#define Re 8.5e6

struct lconv *leng;
char sep,no_sep;


/***FUNCIONES DE LECTURA Y ESCRITURA***/

void lee_float (FILE*ent,float *pf){
   char s[40],*p;
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%f",pf);
   }

void lee_double (FILE*ent,float *pf){
   char s[40],*p;
```

```
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%lf",pf);
   }

void escribe_float (FILE*sal,float *pf){
   char s[40],*p;
   sprintf(s,"%.0f",*pf);
   while (p=strchr(s,no_sep))  *p=sep;
   fprintf(sal,"%s",s);
   }


/*****************MAIN*************/

main (int argc,char *argv[]){
float CalculaAtenuacion(float *perfil, float cellsize, float frec, float K, float
*vector_distancias, int ep, float hr, float R, float Rt, int e1, int ft, int ct, int *y,
int *x);
/*1. Se pasan como argumentos 'ep' (en vez de 'npuntos') y 'hr'; se eliminan como
argumentos 'perfil_flecha' e 'indices_ptoscorte'*/


/***FICHEROS UTILIZADOS***/

FILE *grid;
FILE *fdatos;
FILE *resultados;
FILE *comprueba;
FILE *mdt;
FILE *tiempo;


/***VARIABLES AUXILIARES***/

float pi = 3.141592654;
int i,j;
int fr,cr;
long int prod,ind,prod1,ind1,indtx;
float *mapa,*campo;
char *tira;
char diag[160];
float azimut, elevacion;
float G=0;
float temp;
int dx,dy,e1;
float e;
int *x,*y;
float paso,aux;
```

```
float *perfil;
/*2. Se suprimen los vectores 'perfil_flecha' e 'indices_ptoscorte' para definirlos en
CalculaAtenuacion.cpp*/
float *vector_distancias;
float atenuacion;
float paso2;
int obstaculo;
float k;
int *ind_calc;
/*3. Se define el vector 'ind_calc'*/
int ep;
/*4. Se define el entero 'ep'*/
clock_t start, end;


/***PARAMETROS OBTENIDOS DEL FICHERO DE DATOS***/

float frecuencia;
float ht,hr;
int nfils,ncols;
int ft,ct;
float celda;
float R;
float Rtotal;
int ncols2,nrows2,nodata2;
double xllcorner2,yllcorner2,cellsize2;
int ncols1,nrows1,nodata1;
double xllcorner1,yllcorner1,cellsize1;
int x1,y1;
int filasab,filasar,coli,cold;
int fa,fab,ci,cd;
float radio;
float figura,xllcorner,yllcorner;

start=clock();
leng=localeconv();
sep=*(leng->decimal_point);
if (sep==',')no_sep='.';
else no_sep=',';


/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO DE DATOS***/

frecuencia=atof(argv[5]);
ht=atof(argv[6]);
hr=atof(argv[7]);
nfils=atoi(argv[8]);
ncols=atoi(argv[9]);
ct=atoi(argv[10]);
```

```
ft=atoi(argv[11]);
celda=atof(argv[12]);
obstaculo=atof(argv[13]);
k=atof(argv[14]);
azimut=atof(argv[15]);
elevacion=atof(argv[16]);

figura=atof(argv[18]);
radio=atof(argv[19]);
xllcorner=atof(argv[20]);
yllcorner=atof(argv[21]);


/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO MDT PARA COPIARLOS EN EL FICHERO
RESULTADOS***/

if ((mdt=fopen(argv[4],"r"))!=NULL){

   if ((resultados=fopen(argv[3],"w"))!=NULL){

      tira=(char*)calloc(80,sizeof(char));

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&ncols2);
      fprintf(resultados,"%d\n",ncols2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&nrows2);
      fprintf(resultados,"%d\n",nrows2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&xllcorner2);
      fprintf(resultados,"%lf\n",xllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&yllcorner2);
      fprintf(resultados,"%lf\n",yllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s      ",tira);
      lee_double(mdt,&cellsize2);
      fprintf(resultados,"%lf\n",cellsize2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s  ",tira);
```

```
      fscanf(mdt,"%d\n",&nodata2);
      fprintf(resultados,"%d\n",nodata2);

      fclose(resultados);
      free(tira);
   }

   else{
      printf("No se puede abrir fichero de salida\n");
      exit(1);
   }

   fclose(mdt);
}


/***LECTURA DE LA CABECERA DEL FICHERO ALTURAS***/

if ((grid=fopen(argv[2],"r"))!=NULL){

   tira=(char*)calloc(80,sizeof(char));

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&ncols1);

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nrows1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&xllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&yllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&cellsize1);

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nodata1);

   free(tira);


/***RESERVA DE MEMORIA PARA EL MDT (MAPA) Y EL MAPA DE CAMPO (CAMPO) RESULTADO***/

   prod=(long) nrows1*ncols1;
   mapa=(float*)calloc(prod,sizeof(float));
   indtx=(long)ncols1*ft+ct;
   campo=(float*)calloc(prod,sizeof(float));
```

```
   if (!mapa||!campo){
      printf("No queda memoria disponible para ciudad o campo\n");
      exit(1);
   }

   fa=((yllcorner1+nrows1*cellsize1)-(yllcorner+nfils*cellsize1))/cellsize1;
   fab=nrows1-fa-nfils;
   ci=(xllcorner-xllcorner1)/cellsize1;
   cd=ncols1-ci-ncols;
/*5. Estas cuatro variables se calculan antes que en el caso del código primario para
poder realizar la comprobación siguiente (modificación número 6) */

   for (i=0;i<nrows1;i++)
   for (j=0;j<ncols1;j++){
      ind=(long) ncols1*i+j;
      lee_float(grid,&temp);
      if(fabs(temp+9999)<1)temp=0;

      if (ind==indtx) mapa[ind]=temp+ht;
      else {
         mapa[ind]=temp;
      }

      if ((i>fa-1) && (i<fa+nfils) && (j>ci-1) && (j<ci+ncols)){
/*6. Se fuerza a que para todos los índices que se encuentren dentro del área de
cobertura (los que se desean calcular) el valor de 'campo[ind]' sea igual a -9999*/
         campo[ind]=-9999;
      }

   }

   fclose(grid);
}

else{
   printf("No se puede abrir fichero de entrada\n");
   exit(1);
}


/***BUCLE PRINCIPAL***/

for (fr=0;fr<nrows1;fr++)
for (cr=0;cr<ncols1;cr++){ /***INICIO DEL BUCLE PRINCIPAL***/
   ind=(long)ncols1*fr+cr;

   if(ind!=indtx) mapa[ind]=mapa[ind]+hr;

   if ((mapa[ind]>=hr) (fr>fa-1) && (fr<fa+nfils) && (cr>ci-1) && (cr<ci+ncols) &&
```

```
   ( (fr==fa) || (fr==fa+nfils-1) || (cr==ci) || (cr==ci+ncols-1) ) ){

/***INICIO DE LA CONDICIÓN DE CELDA EN LA QUE SE DESEA HALLAR LAS PÉRDIDAS***/
/*7. Se definen las nuevas condiciones para que sólo se levante el perfil (es decir, se
obtengan los vectores 'perfil', 'vector_distancias' e 'ind_calc') en las celdas de los
bordes del área de cobertura a calcular*/


      /***CÁLCULO DE LA DISTANCIA TX-RX***/

      dx=cr-ct;
      dy=fr-ft;
      e=(float)sqrt(dx*dx+dy*dy);
      R=e*(celda);
      Rtotal=(float)sqrt(R*R+(mapa[indtx]-mapa[ind])*(mapa[indtx]-mapa[ind]));
/*8. Se elimina la condición de devolver -9999 en caso de que figura=1 y R>radio*/

         /***RESERVA DE MEMORIA***/

         e1=floor(e)+1;
         x=(int*)calloc(e1,sizeof(int));
         y=(int*)calloc(e1,sizeof(int));


         /***CREACIÓN DEL VECTOR X***/

         paso=(e==0) ? 0 : dx/floor(e);
         for (i=0;i<(e1-1);i++){
            aux=ct+i*paso-floor(ct+i*paso);
            if (aux<0.5) x[i]=floor(ct+i*paso);
            else x[i]=ceil(ct+i*paso);
         }

         x[e1-1]=cr;


         /***CREACIÓN DEL VECTOR Y***/

         paso=(e==0) ? 0 : dy/floor(e);
         for (i=0;i<(e1-1);i++){
            aux=ft+i*paso-floor(ft+i*paso);
            if (aux<0.5) y[i]=floor(ft+i*paso);
            else y[i]=ceil(ft+i*paso);
         }

         y[e1-1]=fr;


         /***SE HALLA EL PERFIL***/
```

```
        perfil=(float*)calloc(e1,sizeof(float));
/*9.  Se  suprime  la  reserva  de  memoria  para  los  vectores  'perfil_flecha'  e
'indices_ptoscorte'*/
        vector_distancias=(float*)calloc(e1,sizeof(float));
        ind_calc=(int*)calloc(e1,sizeof(int));
/*10.  En  'ind_calc'  se  guardan  los  índices  que  componen  el  perfil  completo  de  la
iteración actual (desde el transistor a cada una de las celdas que se encuentre en el
exterior del área de cobertura)*/

        if (!perfil){
           printf("No queda memoria disponible para perfil\n");
           exit(1);
        }

        for(j=0;j<e1;j++){
           ind=(long) ncols1*y[j]+x[j];
           perfil[j]=mapa[ind];
           ind_calc[j]=ind;
/*11. Se rellena el array 'ind_calc' como se ha comentado anteriormente*/
        }

        perfil[e1-1]=perfil[e1-1]-hr;
/*12.  Se  resta  la  altura  del  receptor  a  'perfil[]',  para  ir  sumándola  y  restándola
sucesivamente en cada iteración dentro del método CalculaAtenuacion*/


        /***CREACIÓN DEL VECTOR DE DISTANCIAS***/

        paso2=(e==0) ? 0 : R/floor(e);
        for (i = 0; i < (e1-1); i++) {
           vector_distancias[i]=i*paso2;
        }

        vector_distancias[e1-1]=R;

        if(ind!=indtx)
        mapa[ind]=mapa[ind]-hr;
/*13. Se resta la altura 'hr' a 'mapa[ind]' para sucesivos levantamientos de perfil*/

        G=0;

        if (e1==1) {
           atenuacion=-9999;
           campo[ind]=atenuacion;
        }

        else{
/***14. BUCLE PARA REALIZAR LA LLAMADA AL MÉTODO CALCULAATENUACION PARA AQUELLOS INDICES
QUE DEBO CALCULAR DENTRO DEL PERFIL COMPLETO DE ESTA ITERACIÓN***/
```

```
            ep=e1-1;
            for(i=(e1-1);i>0;i--){

               if(campo[ind_calc[ep]]==(-9999)){
                  atenuacion=CalculaAtenuacion(perfil,celda,frecuencia,k,
                  vector_distancias,ep,hr,R,Rtotal,e1,ft,ct,y,x);

                  campo[ind_calc[ep]]=atenuacion - G;
               }

               ep=ep-1;
            }

         }

         free(x);
/*15. Se libera el espacio de memoria de los vectores 'x' e 'y' para cada iteración al
igual que el vector 'perfil', 'vector_distancias' e 'ind_calc'*/
         free(y);
         free (perfil);
         free (vector_distancias);
/*16. De nuevo se elimina la liberación de memoria de los vectores 'indices_ptoscorte' y
'perfil_flecha'*/
         free (ind_calc);
/*17. Se liberan los índices de este perfil para volver a rellenarlo en la siguiente
iteración*/
/*18. Se elimina la llave que cierra el bucle else abierto para el cálculo de pérdidas
en caso de que no se cumpla la condición ya suprimida en el comentario número 8*/
   } /***FIN DE LA CONDICIÓN DE CELDA EN LA QUE HEMOS CALCULADO LAS PÉRDIDAS***/

   else if ((fr<fa) || (fr>fa+nfils-1) || (cr<ci) || (cr>ci+ncols-1)){
/*19. Con esta condición, se asigna valor 'campo[ind]' a todas las celdas que estén
fuera del área de cobertura*/
      campo[ind]=-9999;
   }

   if(ind!=indtx) mapa[ind]=mapa[ind]-hr;
/*20. Se resta la altura del receptor a todas aquellas celdas que no pertenezcan al área
de cobertura seleccionada*/
} /***FIN DEL BUCLE PRINCIPAL***/


/***ESCRITURA DEL FICHERO DE CAMPO EN DISCO***/

if ((resultados=fopen(argv[3],"a"))!=NULL){

   filasar=((yllcorner2+cellsize2*nrows2)-(yllcorner1+cellsize1*nrows1))/
   cellsize2;
   filasab=nrows2-nrows1-filasar;
```

```
   coli=(xllcorner1-xllcorner2)/cellsize2;
   cold=ncols2-ncols1-coli;

   for (x1=0;x1<filasar;x1++){
      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   for (i=0;i<nrows1;i++){
      for (y1=0;y1<coli;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      for (j=0;j<ncols1;j++){
         ind1=(long) ncols1*i+j;
         escribe_float(resultados,&(campo[ind1]));
         fprintf(resultados," ");
      }

      for (y1=0;y1<cold;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   for (x1=0;x1<filasab;x1++){
      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   fclose (resultados);
   printf("El campo está  en disco\n");
}

else{
   printf("No se puede abrir fichero de salida\n");
   exit (1);
}
```

```
free(campo);
free(mapa);

comprueba=fopen(argv[22],"w");
fprintf(comprueba,"\n");

fclose (comprueba);
end = clock();

if ((tiempo=fopen("c:\\arcgis\\RAGIS\\tiempo.txt","w"))!=NULL){
   fprintf(tiempo,"The time was: %f\n", (end - start) / CLK_TCK);
   fclose(tiempo);
}

} /***FIN DE MAIN***/
```

3.3.2. Fichero calculaatenuacion.cpp.

Como se ha visto en el apartado 1.2.2., este fichero consta de cinco métodos: *CalculaPerfil_flecha*, *LOS*, *CalculaFuniculo*, *espacio_libre* y *CalculaAtenuacion*. Sólo en este último se han realizado modificaciones, así que para evitar información redundante, se evitará mostrar en este punto los cuatro primeros métodos.

```
/***5.- MÉTODO CALCULAATENUACIÓN***/

float CalculaAtenuacion(float *perfil, float cellsize, float frec, float K, float
*vector_distancias, int ep, float hr, float R, float Rt, int e1, int ft, int ct, int *y,
int *x) {
/*21. Se pasa como argumento 'ep' en vez de npuntos, y también 'hr', puesto que ahora el
sumar y restar la altura de la antena receptora no se hace en principal.cpp, sino aquí;
además se elimina 'perfil_flecha' e 'indices_ptos corte'*/

   float *flecha;
   float Ltotal=0, LD=0, L0=0, Llluvia=0, Lgases=0, Ldi=0, sumatorioLD=0;
   int vision;
   float roo = 7.5;
   float tasa = 25;
   char polarizacion = 'h';
   int i, j;
   float *alturas_obs;
   float *distancias_obs;
   int num_vanos;
   float d1, d2, vi, hi, tang, vimax;
   float sumatorioLsdi=0, Lsdi=0, Lsd_max=0;
```

```
   double Cn=0;
   double a, b, c;
   float htx, hrx, posicion_tx, posicion_rx;
   int long_ptos,l;
   float *perfil_flecha;
/*22.  Además  es  aquí  donde  se  declaran  los  vectores  'perfil_flecha'  e
'indices_ptoscorte', los que también se irán modificando con cada llamada a método*/
   int *indices_ptoscorte;
   float *perfil_actual;
/*23. Se declara el nuevo vector 'perfil_actual' a reutilizar en cada llamada a método*/
   float *vector_distancias_actual;
/*24. Se declara el nuevo vector 'vector_distancias_actual' a reutilizar en cada llamada
a método*/
   float dist_actual;
/*25. Se declara 'dist_actual' y 'paso_actual' para crear el 'vector_distancias_actual'
adecuado cuando no se calculen las pérdidas en celdas que no estén en el exterior del
área de cobertura*/
   float paso_actual;
   int ep1;
/*26. Se declara 'ep1'*/

   ep1=ep+1;
/*27. 'ep1' es lo mismo que el 'e1', pero para cada una de las iteraciones de 'ep'*/

   flecha = (float*)calloc(ep1,sizeof(float));
   alturas_obs = (float*)calloc(ep1,sizeof(float));
   distancias_obs = (float*)calloc(ep1,sizeof(float));
   perfil_actual = (float*)calloc(ep1,sizeof(float));
/*28. Se reserva memoria para los nuevos vectores*/
   vector_distancias_actual = (float*)calloc(ep1,sizeof(float));
   perfil_flecha = (float*)calloc(ep1,sizeof(float));
   indices_ptoscorte = (int*)calloc(ep1,sizeof(float));

   if (K == (float)0){
      return -1;
   }

   if (ep1==e1){
/*29. Creación de los vectores 'perfil_actual' y 'vector_distancias_actual' para las
celdas exteriores del área de cobertura a calcular*/
      for (i=0; i<ep1; i++){
         perfil_actual[i]=perfil[i];
         vector_distancias_actual[i]=vector_distancias[i];
      }

      perfil_actual[ep]=perfil_actual[ep]+hr;
   }

   else{
```

```
/*30. Creación de los vectores 'perfil_actual' y 'vector_distancias_actual' para las
celdas interiores del área de cobertura a calcular*/
      dist_actual=sqrt((pow((y[ep]-ft)*cellsize,2))+(pow((x[ep]-ct)*cellsize,2)));
      paso_actual=dist_actual/ep;
      for (i=0; i<ep1; i++){
         perfil_actual[i]=perfil[i];
         vector_distancias_actual[i]=i*paso_actual;
      }

      perfil_actual[ep]=perfil_actual[ep]+hr;
      R=vector_distancias_actual[ep];
/*31. Además del vector de las distancias, hay que modificar 'R' y 'Rt'*/
      Rt=(float)sqrt(R*R+(perfil_actual[0]-perfil_actual[ep])*(perfil_actual[0]
      -perfil_actual[ep]));
   }

   if ((figura==1) && (R > radio)){
/*32. Bucle if con el que se devuelve -9999 y se libera la memoria reservada con calloc
al principio de la llamada, en caso que se quiera calcular un área de cobertura
circular, y dicha altura del MDT no pertenezca a esa circunferencia de cálculo*/
      free(distancias_obs);
      free(alturas_obs);
      free(flecha);
      free(perfil_actual);
      free(vector_distancias_actual);
      free(perfil_flecha);
      free(indices_ptoscorte);

      return -9999;

   }


   /***LLAMADA AL MÉTODO CALCULAPERFIL_FLECHA***/

   CalculaPerfil_flecha(perfil_actual, ep1, K, cellsize, perfil_flecha);
/*33. Se pasan como parámetros 'perfil_actual' y 'ep1'*/
   for (i = 0; i<ep1; i++)  flecha[i] = perfil_flecha[i]-perfil_actual[i];
/*34. La condición del bucle for es hasta ep1, y a 'perfil_flecha' restamos
'perfil_actual'*/

////////////////////////////////////ATENUACIÓN/////////////////////////////////////

   R=R/1000;
   Rt=Rt/1000;


   /***LLAMADA AL MÉTODO ESPACIO_LIBRE***/
```

```
   L0 = espacio_libre(frec, Rt);


/***LLAMADA AL MÉTODO LOS (devuelve un 1 si hay vision directa y un 0 si no la hay)***/

   vision = LOS(vector_distancias_actual, perfil_flecha, ep1);
/*35. Se pasan como argumentos 'vector_distancias_actual' y 'ep1'*/

////////////////////////////////////CASO NLOS/////////////////////////////////////

   if (vision == 0){


      /***LLAMADA AL MÉTODO CALCULAFUNICULO***/

      long_ptos = CalculaFuniculo(perfil_flecha, ep1, cellsize,
      vector_distancias_actual, indices_ptoscorte);
/*36. Se pasan como argumentos 'ep1' y 'vector_distancias_actual' ('long_puntos'
devuelve el número de puntos que conforman el polígono funicular)*/

///////////////PÉRDIDAS POR DIFRACCIÓN Y DESPEJAMIENTO INSUFICIENTE///////////////

      for(i=0; i<ep1; i++) vector_distancias_actual[i]=vector_distancias_actual[i]/1000;
/*37. La condición es hasta 'ep1', y se pasa a km los valores del
'vector_distancias_actual'*/

      for (i = 0; i<long_ptos; i++){
         alturas_obs[i] = perfil_flecha[indices_ptoscorte[i]];
         distancias_obs[i] = vector_distancias_actual[indices_ptoscorte[i]];
/*38. Se trabaja con 'vector_distancias_actual'*/
      }

      num_vanos=long_ptos-2; //NÚMERO DE VANOS (QUITANDO TX Y RX)
      sumatorioLD=0;

      //SE CALCULAN EN PRIMER LUGAR LAS PÉRDIDAS POR DIFRACCIÓN DEBIDAS A CADA VANO

      for (i = 1; i<=num_vanos; i++){
         d1 = distancias_obs[i]-distancias_obs[i-1];
         d2 = distancias_obs[i+1]-distancias_obs[i];
         tang = (alturas_obs[i+1]-alturas_obs[i-1])/(d1+d2);
         hi = tang*d1+alturas_obs[i-1]-alturas_obs[i];
         hi = -hi;
         vi = 2.58*pow(10,-3)*hi*sqrt(frec*((d1+d2)/(d1*d2)));
         Ldi = 6.9+20*log10(sqrt(pow((vi-0.1),2)+1)+vi-0.1);
         sumatorioLD = sumatorioLD+Ldi;
      } //FIN DEL FOR

      //SE CALCULAN LAS PÉRDIDAS POR DESPEJAMIENTO INSUFICIENTE
```

```
      Lsd_max=sumatorioLsdi=0;

      for (i=0; i<long_ptos-1; i++){

         if(indices_ptoscorte[i]!=indices_ptoscorte[i+1]-1){
            vimax=-100000;
            l= indices_ptoscorte[i+1]-indices_ptoscorte[i]+1;
            if(l<=3) Lsdi=Lsd_max=0;
            else{
               for (j=indices_ptoscorte[i]+1; j<=indices_ptoscorte[i+1]-1; j++){

                  d1=vector_distancias_actual[j]-vector_distancias_actual
                  [indices_ptoscorte[i]];
/*39. Para 'd1' y 'd2' se trabaja con 'vector_distancias_actual'*/


                  d2=vector_distancias_actual [indices_ptoscorte[i+1]]-
                  vector_distancias_actual [j];

                  tang=(perfil_flecha[indices_ptoscorte[i+1]]-perfil_flecha
                  indices_ptoscorte[i]])/(d1+d2);

                  hi=tang*d1+perfil_flecha[indices_ptoscorte[i]]-perfil_flecha[j];
                  hi=-hi;
                  vi = 2.58*pow(10,-3)*hi*sqrt(frec*((d1+d2)/(d1*d2)));
                  if(vi>vimax) vimax=vi;
               }

               if(vimax>-1) Lsdi=6.9+20*log10(sqrt(pow((vimax-0.1),2)+1)+vimax-0.1);
               else Lsdi = 0;

               if(Lsdi>Lsd_max) Lsd_max=Lsdi;
            }

         }

      } //FIN DEL FOR

      //CÁLCULO DEL FACTOR DE CORRECCIÓN CN

      if(num_vanos>1){ //SE CALCULA EL FACTOR DE CORRECCIÓN SÓLO SI EXISTE MÁS DE UN
                       //OBSTÁCULO

         a=1;
         b=0;
         c=1;

         for (i=0; i<long_ptos; i++){
            if(i<long_ptos-3){
```

```
            a=a*(distancias_obs[i+2]-distancias_obs[i+1]);
         }

         b=distancias_obs[long_ptos-1];
         if (i<long_ptos-2){
            c=c*(distancias_obs[i+2]-distancias_obs[i]);
         }

      }

      Cn=(double)(a*b)/c;

      //PÉRDIDAS DE DIFRACCIÓN LD

      LD = sumatorioLD + Lsd_max - 10*log10(Cn);
   }
   else{
      LD = sumatorioLD + Lsd_max;
   }

   Ltotal=LD+L0;
} //FIN DEL IF
////////////////////////////////////CASO LOS/////////////////////////////////////

else if (vision == 1){

   //PÉRDIDAS POR DIFRACCIÓN
   LD = 0;

   //PÉRDIDAS POR DESPEJAMIENTO INSUFICIENTE
   for(i = 0; i<ep1; i++) vector_distancias_actual[i]=vector_distancias_actual[i]
   /1000;
/*40.  La  condición  del  bucle  for  es  hasta  'ep1',  y  se  trabaja  con
'vector_distancias_actual'*/

   htx=perfil_flecha[0];
   hrx=perfil_flecha[ep1-1];
/*41. A 'hrx' le pertenece la posición 'ep1-1'*/
   posicion_tx= vector_distancias_actual[0];
/*42. Se trabaja con 'vector_distancias_actual'*/
   posicion_rx= vector_distancias_actual[ep1-1];
/*43. Se trabaja con 'vector_distancias_actual' y 'ep1'*/

   Lsd_max=0;

   l=ep1;
/*44. 'l' es igual a 'ep1'*/
   if(l<=3) Lsdi=Lsd_max=0;
   else{
```

```
         for (i=1; i<=ep1-2;i++){
/*45. Condición hasta 'ep1-2'*/
            d1=vector_distancias_actual[i]-posicion_tx;
/*46. Para 'd1' y 'd2' se trabaja con 'vector_distancias_actual'*/
            d2=posicion_rx - vector_distancias_actual[i];
            tang=(hrx-htx)/R;
            hi=tang*d1+htx-perfil_flecha[i];
            hi=-hi;
            vi=2.58e-3*sqrt((frec*(d1+d2))/(d1*d2))*hi;
            if(vi>-1) Lsdi=6.9+20*log10(sqrt(pow((vi-0.1),2)+1)+vi-0.1);
            else Lsdi =0;
            if(Lsdi>Lsd_max) Lsd_max=Lsdi;
         }

      } //FIN DEL ELSE

      Ltotal=LD+L0+Lsd_max;
   } //FIN DEL ELSE IF

   free(distancias_obs);
   free(alturas_obs);
   free(flecha);
   free(perfil_actual);
/*47. Se libera la memoria de los cuatro vectores nuevos utilizados en cada llamada
(para cada 'ep' diferente de cada 'e1' total)*/
   free(vector_distancias_actual);
   free(perfil_flecha);
   free(indices_ptoscorte);

   return  Ltotal;
}
/*48. LLave que cierra la condición else de que no se está cumpliendo que figura=1 y
R>radio*/

}
```

3.4. MODIFICACIÓN Y PARTICULARIDADES DEL MODELO RURAL.

En este apartado se explica cada uno de los cambios realizados sobre el código primario. Hay que destacar que tanto para el proyecto rural, como para el urbano, las modificaciones son similares, puesto que se centran principalmente en el **levantamiento del perfil**, aunque habrá que tener muy en cuenta también las particularidades de cada uno de los modelos.

En el momento que se ejecuta el proyecto "UITR-526.ide", comienza a funcionar el archivo "principal.cpp"; a partir de éste se van realizando una serie de llamadas a distintos métodos que se encuentran en el archivo "calculaatenuacion.cpp", y que irán devolviendo una serie de valores para que el método *main* ("principal.cpp") pueda seguir trabajando hasta conseguir como resultado el cálculo de cobertura de una determinada área (ya sea rectangular o circular) seleccionada.

3.4.1. Modificación en el fichero principal.cpp.

El código comienza (a partir de "principal.cpp", como se acaba de comentar) con unos determinados *include* para vincular las librerías necesarias donde se encuentran las funciones a utilizar en adelante.

```
#include <time.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <locale.h>

#define PI abs(acos(-1.0))
#define Re 8.5e6
```

Las siguientes líneas son una serie de métodos que actúan como funciones de lectura y escritura para que no exista problemas con los datos si estos están formados por puntos o por comas.

```
struct lconv *leng;
char sep,no_sep;

/***FUNCIONES DE LECTURA Y ESCRITURA***/

void lee_float (FILE*ent,float *pf){
   char s[40],*p;
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%f",pf);
   }

void lee_double (FILE*ent,float *pf){
   char s[40],*p;
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%lf",pf);
   }

void escribe_float (FILE*sal,float *pf){
   char s[40],*p;
   sprintf(s,"%.0f",*pf);
   while (p=strchr(s,no_sep))  *p=sep;
   fprintf(sal,"%s",s);
   }
```

A continuación se presentan las definiciones de todas y cada una de las variables que se van a utilizar en el fichero "principal.cpp" del proyecto "UITR-526.ide". La primera parte de este bloque está formada por la apertura del *main* y la definición del método *CalculaAtenuacion* que se encuentra en el fichero "calculaatenuacion.cpp".

Tras ello, se definen los ficheros que se van a crear a medida que transcurra la ejecución, a la vez que todas las variables a utilizar.

```
/*****************MAIN*************/

main (int argc,char *argv[]){
float CalculaAtenuacion(float *perfil, float cellsize, float frec, float K, float
*vector_distancias, int ep, float hr, float R, float Rt, int e1, int ft, int ct, int *y,
int *x);
/*1. Se pasan como argumentos 'ep' (en vez de 'npuntos') y 'hr'; se eliminan como
argumentos 'perfil_flecha' e 'indices_ptoscorte'*/


/***FICHEROS UTILIZADOS***/

FILE *grid;
FILE *fdatos;
FILE *resultados;
FILE *comprueba;
FILE *mdt;
FILE *tiempo;


/***VARIABLES AUXILIARES***/

float pi = 3.141592654;
int i,j;
int fr,cr;
long int prod,ind,prod1,ind1,indtx;
float *mapa,*campo;
char *tira;
char diag[160];
float azimut, elevacion;
float G=0;
float temp;
int dx,dy,e1;
float e;
int *x,*y;
float paso,aux;
float *perfil;
/*2. Se suprimen los vectores 'perfil_flecha' e 'indices_ptoscorte' para definirlos en
CalculaAtenuacion.cpp*/
float *vector_distancias;
float atenuacion;
float paso2;
int obstaculo;
float k;
int *ind_calc;
/*3. Se define el vector 'ind_calc'*/
int ep;
/*4. Se define el entero 'ep'*/
clock_t start, end;
```

```
/***PARAMETROS OBTENIDOS DEL FICHERO DE DATOS***/

float frecuencia;
float ht,hr;
int nfils,ncols;
int ft,ct;
float celda;
float R;
float Rtotal;
int ncols2,nrows2,nodata2;
double xllcorner2,yllcorner2,cellsize2;
int ncols1,nrows1,nodata1;
double xllcorner1,yllcorner1,cellsize1;
int x1,y1;
int filasab,filasar,coli,cold;
int fa,fab,ci,cd;
float radio;
float figura,xllcorner,yllcorner;

start=clock();
leng=localeconv();
sep=*(leng->decimal_point);
if (sep==',')no_sep='.';
else no_sep=',';
```

Para finalizar, en el último párrafo de este bloque, se activa la variable *clock* que dará el tiempo de cálculo de la cobertura al final de la ejecución.

Como se ha podido apreciar, hay nuevas variables definidas, y otras que ya no se definen aquí, sino en el método *CalculaAtenacion*, pero eso se explicará más adelante, cuando estas variables jueguen un papel significativo.

El siguiente paso es tomar como argumentos la serie de parámetros introducidos a partir del entorno gráfico de *RADIOGIS*, que determinan una importante cadena de datos para el cálculo de la cobertura deseada: emplazamiento de la antena transmisora y receptora en número de celdas en función del grid (*ht*, *hr* y *ct*,*cr* respectivamente), tamaño de la celda en metros (*celda*), altura del transmisor y del receptor en metros (*ht* y *hr* respectivamente), área de cobertura cuadrada o circular (*figura*), coordenadas del extremo inferior del área de cobertura en metros (*xllcorner* e *yllcorner*), así como el número de filas y columnas que dicha área abarca (*nfils* y *ncols* respectivamente), radio

en metros a partir del transmisor del área de cobertura en caso de que ésta sea circular (*radio*), etc).

```
/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO DE DATOS***/

frecuencia=atof(argv[5]);
ht=atof(argv[6]);
hr=atof(argv[7]);
nfils=atoi(argv[8]);
ncols=atoi(argv[9]);
ct=atoi(argv[10]);
ft=atoi(argv[11]);
celda=atof(argv[12]);
obstaculo=atof(argv[13]);
k=atof(argv[14]);
azimut=atof(argv[15]);
elevacion=atof(argv[16]);

figura=atof(argv[18]);
radio=atof(argv[19]);
xllcorner=atof(argv[20]);
yllcorner=atof(argv[21]);
```

Acto seguido, se copian los parámetros del fichero que contiene el MDT (mapa digital del terreno sobre el que se va a trabajar), en el fichero que irá almacenando los resultados de las pérdidas en el receptor.

```
/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO MDT PARA COPIARLOS EN EL FICHERO
RESULTADOS***/

if ((mdt=fopen(argv[4],"r"))!=NULL){

   if ((resultados=fopen(argv[3],"w"))!=NULL){

      tira=(char*)calloc(80,sizeof(char));

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&ncols2);
      fprintf(resultados,"%d\n",ncols2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&nrows2);
```

```
      fprintf(resultados,"%d\n",nrows2);
      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&xllcorner2);
      fprintf(resultados,"%lf\n",xllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&yllcorner2);
      fprintf(resultados,"%lf\n",yllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s      ",tira);
      lee_double(mdt,&cellsize2);
      fprintf(resultados,"%lf\n",cellsize2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s  ",tira);
      fscanf(mdt,"%d\n",&nodata2);
      fprintf(resultados,"%d\n",nodata2);

      fclose(resultados);
      free(tira);
   }

   else{
      printf("No se puede abrir fichero de salida\n");
      exit(1);
   }

   fclose(mdt);
}
```

Los datos copiados, por ejemplo, para el MDT de la región de Murcia con resolución de 30 metros son (**ejemplo explicativo 2**):

ncols2	→	**6034**
nrows2	→	**5426**
xllcorner2	→	**550985**
yllcorner2	→	**4135236**
cellsize2	→	**30**
nodata2	→	**-9999**

Estos parámetros aportan la información básica de las características del MDT: *ncols* y *nrows* dan a conocer su número de filas y columnas, *xllcorner* e *yllcorner* reflejan las coordenadas cartográficas en metros del extremo inferior izquierdo del mapa, con *cellsize* se conoce la separación en metros entre celdas, y por último, cuando en el fichero de resultados aparezca el valor *-9999*, significará que es un dato sin valor, es decir, en esa celda no se ha calculado las pérdidas al no pertenecer al área de cobertura de cálculo seleccionada.

El siguiente paso es igual al anterior, pero ahora, en lugar de copiar los parámetros de la cabecera del fichero del MDT en el fichero de resultados, se almacenan esos parámetros en variables, las cuales luego serán utilizadas durante transcurso del programa.

```
/***LECTURA DE LA CABECERA DEL FICHERO ALTURAS***/

if ((grid=fopen(argv[2],"r"))!=NULL){

   tira=(char*)calloc(80,sizeof(char));

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&ncols1);

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nrows1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&xllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&yllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&cellsize1);

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nodata1);

   free(tira);
```

Los valores que toman las variables mencionadas son los siguientes (igual que anteriormente, pero con diferentes nombres, que serán utilizados a partir de este momento):

ncols1	→	**6034**
nrows1	→	**5426**
xllcorner1	→	**550985**
yllcorner1	→	**4135236**
cellsize1	→	**30**
nodata1	→	**-9999**

A continuación, se calcula el número total de celdas de las que consta la matriz MDT (*prod*), y entre todas ellas, cuál es el índice del transmisor (*indtx*). Además, se reserva con la función *calloc* el espacio de memoria necesario para almacenar las alturas del MDT (*mapa*), y los resultados de las pérdidas calculadas (*campo*). Cada una de estas reservas contendrá un número de bloques igual al número de celdas dado por *prod*, y cada uno de esos bloques tendrá el tamaño en bits de un dato del tipo *float* (32 bits).

El bucle *if* del final aborta la ejecución del programa en caso que no haya memoria disponible para almacenamiento de la variable *campo* o *mapa*.

```
/***RESERVA DE MEMORIA PARA EL MDT (MAPA) Y EL MAPA DE CAMPO (CAMPO) RESULTADO***/

   prod=(long) nrows1*ncols1;
   mapa=(float*)calloc(prod,sizeof(float));
   indtx=(long)ncols1*ft+ct;
   campo=(float*)calloc(prod,sizeof(float));

   if (!mapa||!campo){
      printf("No queda memoria disponible para ciudad o campo\n");
      exit(1);
   }
```

Las siguientes variables, obtenidas a partir del cálculo con los parámetros sacados anteriormente de la cabecera del MDT, serán explicadas un poco más adelante,

cuando sean utilizadas para delimitar el área de cálculo de cobertura a partir de una serie de precisas comparaciones y restricciones .

El hecho de que se encuentren a esta altura del código, es para poder realizar la comprobación que corresponde a la modificación número 6, la cual se verá en breve.

```
   fa=((yllcorner1+nrows1*cellsize1)-(yllcorner+nfils*cellsize1))/cellsize1;
   fab=nrows1-fa-nfils;
   ci=(xllcorner-xllcorner1)/cellsize1;
   cd=ncols1-ci-ncols;
/*5. Estas cuatro variables se calculan antes que en el caso del código primario para
poder realizar la comprobación siguiente (modificación número 6) */
```

En el siguiente bloque se realiza el proceso necesario para almacenar en el array unidimensional llamado *mapa* (por el que previamente se había reservado espacio de memoria), todas las alturas correspondientes al fichero del MDT, con la única diferncia respecto a éste de haber sumado la altura de la antena transmisora (*ht*) en el índice en que se encuentre el mencionado transmisor (*indtx*).

```
   for (i=0;i<nrows1;i++)
   for (j=0;j<ncols1;j++){
      ind=(long) ncols1*i+j;
      lee_float(grid,&temp);
      if(fabs(temp+9999)<1)temp=0;

      if (ind==indtx) mapa[ind]=temp+ht;
      else {
         mapa[ind]=temp;
      }

      if ((i>fa-1) && (i<fa+nfils) && (j>ci-1) && (j<ci+ncols)){
/*6. Se fuerza a que para todos los índices que se encuentren dentro del área de
cobertura (los que se desean calcular) el valor de 'campo[ind]' sea igual a -9999*/
         campo[ind]=-9999;
      }

   }

   fclose(grid);
}
else{
   printf("No se puede abrir fichero de entrada\n");
```

```
   exit(1);
}
```

Para ello se recorren todas las filas y columnas de la matriz MDT, asignando la altura correspondiente a cada índice, excepto si se cumple que *ind* = *indtx*, en cuyo caso, como se ha comentado anteriormente se le sumará la altura del transmisor *ht*. En la optimación se aplica una diferencia importante al código, y es que una vez realizadas estas acciones, se introduce una condición mediante *if*, donde se aplican las restricciones necesarias para delimitar el área de cobertura que se ha pasado con el fin de que se calculen las pérdidas en ella; en caso de que el índice pertenezca a dicha área, se le asignará el valor -9999 al array *campo* en dicho índice.

Este paso, como se verá más adelante, es imprescindible para saber en cada momento si las pérdidas han sido calculadas en dicho índice, una vez que se proceda a crear los perfiles, y el mismo índice forme parte de más de uno de esos perfiles levantados.

Finalmente, se cierra el archivo *grid*, seguido de la llave de cierre de la condición *if ((grid=fopen(argv[2],"r"))!=NULL)*.

El *else* último pertenece a dicho *if*, y se ejecutaría en caso de que no se pudiese abrir el fichero que contiene al MDT.

En el siguiente apartado da comienzo el bucle principal del cálculo de pérdidas. Lo primero es recorrer cada una de las celdas del MDT. Con el primer *for* se recorren las filas, y dentro de cada fila, con el siguiente *for*, se recorren las columnas. De tal manera, con la fila y columna identificada, se calcula el índice correspondiente (*ind*), y se comprueba que sea diferente al del transmisor (*indtx*), en cuyo caso se suma la altura del receptor (*hr*). Así pues, la primera iteración será para la fila 0 y columna 0, la siguiente para fila 0 y columna 1, y así sucesivamente hasta llegar a la fila = *nrows1-1* y la columna = *ncols1-1*.

```
/***BUCLE PRINCIPAL***/

for (fr=0;fr<nrows1;fr++)
for (cr=0;cr<ncols1;cr++){    /***INICIO DEL BUCLE PRINCIPAL***/
   ind=(long)ncols1*fr+cr;

   if(ind!=indtx) mapa[ind]=mapa[ind]+hr;
```

La modificación más significativa hasta el momento es la que sucede a continuación.

```
   if ((mapa[ind]>=hr) && (fr>fa-1) && (fr<fa+nfils) && (cr>ci-1) && (cr<ci+ncols) &&
   ( (fr==fa) || (fr==fa+nfils-1) || (cr==ci) || (cr==ci+ncols-1) ) ){

/***INICIO DE LA CONDICIÓN DE CELDA EN LA QUE SE DESEA HALLAR LAS PÉRDIDAS***/
/*7. Se definen las nuevas condiciones para que sólo se levante el perfil (es decir, se
obtengan los vectores 'perfil', 'vector_distancias' e 'ind_calc') en las celdas de los
bordes del área de cobertura a calcular*/
```

Tiene lugar en la condición que delimita si la celda que se está tratando en la actual iteración se encuentra dentro del área de cobertura que se desea calcular; para ello se emplean las siguientes condiciones, de las cuales, para que el *if* de un resultado afirmativo, **se deben cumplir todas y cada una de ellas**:

- *mapa [ind] >= hr*
- *fr > fa – 1*
- *fr < fa + nfils*
- *cr > ci – 1*
- *cr < ci + ncols*

Hasta aquí la condición es similar a la del modelo primario. En la optimización se han añadido otra serie de condiciones, entre las que al menos **es obligatorio el cumplimiento de al menos una de ellas** para proceder al levantamiento del perfil y cálculo de pérdidas. Con estas condiciones se comprueba que la celda se encuentra en uno de los bordes del rectángulo pasado como argumento.

- *fr = fa*
- *fr = fa + nfils – 1*
- *cr = ci*
- *cr = ci + ncols - 1*

Solamente se podrían cumplir dos de estas condiciones a la vez, si la celda estuviese situada en una de las cuatro esquinas, en cuyo caso formaría parte de una de las filas del borde del cuadrante, y de una de sus columnas.

Como nota aclaratoria del funcionamiento del programa en este punto de la condición, se mostrará un ejemplo numérico real (**ejemplo explicativo 3**). Se parte de los siguientes valores previos, los cuales ya han sido explicados:

ncols1	301
nrows1	221
xllcorner1	3650672
yllcorner1	5311282
cellsize1	210

Parámetros de la cabecera del fichero MDT.

frecuencia	900
ht	10
hr	2
nfils	5
ncols	6
ct	6
ft	4
celda	210
obstaculo	0
k	1,33
azimut	0
elevacion	0
figura	0
radio	0
xllcorner	3651512
yllcorner	5356222

Parámetros introducidos como argumentos a partir del entorno gráfico de *RADIOGIS*, para el cálculo de la cobertura deseada.

Tablas 3.1. Ejemplo explicativo 3. Variables iniciales.

Se vuelve a hacer referencia ahora a esa porción del código cuya expliación se había pospuesto hasta este momento.

```
fa=((yllcorner1+nrows1*cellsize1)-(yllcorner+nfils*cellsize1))/cellsize1;
fab=nrows1-fa-nfils;
ci=(xllcorner-xllcorner1)/cellsize1;
cd=ncols1-ci-ncols;
```

Con los datos anteriormente mostrados, los valores de las variables obtenidas son los siguientes:

fa	2
fab	214
ci	4
cd	291

Tabla 3.2. Ejemplo explicativo 3. Variables que delimitan el área de cobertura a calcular.

Estas variables ubican en forma de número de celdas la situación del cuadrante de cobertura en relación con los bordes del MDT completo. De tal manera, se sabe que fuera del cuadrante quedarán 2 filas hacia arriba y 214 hacia abajo; entre ellas se encontrarán las 5 filas que conforman dicho rectángulo. La suma de todas estas filas dan como resultado las 221 que conforman al MDT.

De igual forma, a la izquierda de las 6 columnas del cuadrante existirán 4 columnas, y a su derecha 291, dando como resultado las 301 totales. Vease gráficamente:

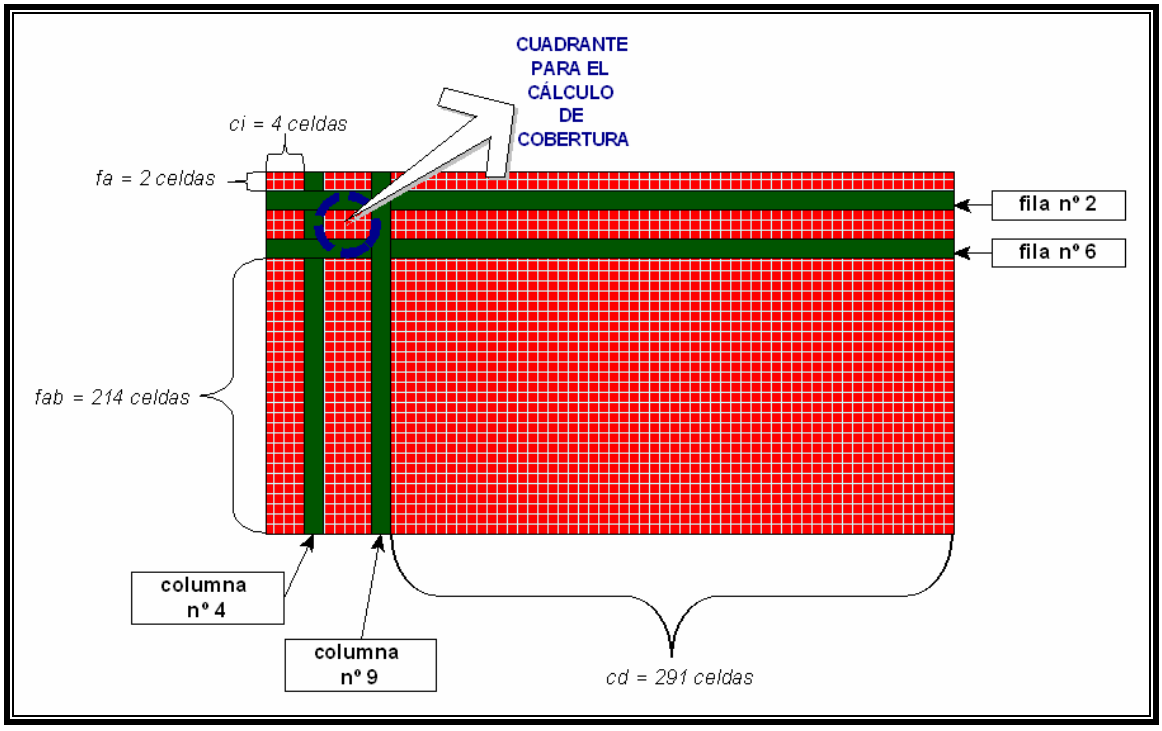


Figura 3.5. Ejemplo explicativo 3. Situación en número de celdas del transmisor y el área de cobertura.

Posteriormente se procede a conocer los datos del receptor, cuya fila (*fr*) y columna (*cr*) dentro de la matriz del MDT, dependen de la iteración en que se encuentre la ejecución del programa. Con esos dos valores se calcula el índice de la celda en que está ubicado el receptor (*ind*), y a partir de esto, se sabrá el valor de la altura de *mapa[ind]*, al que se le sumará la altura física del receptor.

fr	3
cr	9
ind	ncols1 * fr + cr = 301 * 3 + 9 = 912
mapa [912]	440

⇒ Datos del receptor de la presente iteración.

Tabla 3.3. Ejemplo explicativo 3. Variables del receptor.

Con todos estos datos, se llega a la conocida e importante condición. Se va a proceder a la explicación de cada uno de sus términos con la ayuda extra del ejemplo numérico.

```
if ((mapa[ind]>=hr) (fr>fa-1) && (fr<fa+nfils) && (cr>ci-1) && (cr<ci+ncols) &&
( (fr==fa) || (fr==fa+nfils-1) || (cr==ci) || (cr==ci+ncols-1) ) ){
```

CONDICIÓN		EXPLICACIÓN
CONDICIONES PRIMERAS. SE HAN DE CUMPLIR TODAS ELLAS		
Mapa [ind] ≥ hr	**440 ≥ 2**	La altura de la celda actual (terreno más física del receptor) es mayor que únicamente la física del receptor.
fr > fa – 1	**3 > 2 - 1**	La fila del receptor no pertenece a las que están por encima del cuadrante de cobertura.
fr < fa + nfils	**3 < 2 + 5**	La fila del receptor no pertenece a las que están por debajo del cuadrante de cobertura.
cr > ci – 1	**9 > 4 - 1**	La columna del receptor no pertenece a las que están a la izquierda del cuadrante de cobertura.
cr < ci + ncols	**9 < 4 + 6**	La columna del receptor no pertenece a las que están a la derecha del cuadrante de cobertura.
CONDICIONES ADICIONALES. SE HA DE CUMPLIR AL MENOS UNA DE ELLAS		
fr = fa	3 ≠ 2	La fila del receptor pertenece a la fila del borde superior del cuadrante de cobertura
fr = fa + nfils – 1	3 ≠ 2 + 5 - 1	La fila del receptor pertenece a la fila del borde inferior del cuadrante de cobertura
cr = ci	9 ≠ 4	La columna del receptor pertenece a la columna del borde izquierdo del cuadrante de cobertura
cr = ci + ncols – 1	**9=4+6-1**	La columna del receptor pertenece a la columna del borde derecho del cuadrante de cobertura

Tabla 3.4. Ejemplo explicativo 3. Condición restrictiva para el levantamiento del perfil.

Si el índice de la actual iteración cumple la condición, se **encontrará en una de las celdas que conforman el borde del rectángulo que se pasa como argumento para el cálculo de pérdidas**.

Nota: un índice puede cumplir como máximo dos condiciones del segundo bloque, y será cuando se encuentre en una de las cuatro esquinas que forman al cuadrante de cobertura (se cumplirá la condición para una columna y una fila del cuadrante).

En caso de que el índice de la iteración no cumpla dicha condición, no se procederá al levantamiento del perfil ni posterior cálculo de pérdidas; sino que se pasará a la siguiente iteración, para volver a realizar la comprobación de dicha condición con los nuevos valores en ese caso.

Si la respuesta a la condición es afirmativa (como en el caso el ejemplo explicativo 3) se procederá a levantar el perfil desde ese índice hasta el transmisor. Para comenzar con este proceso, lo primero será calcular la distancia entre el transmisor y el receptor.

```
        /***CÁLCULO DE LA DISTANCIA TX-RX***/

    dx=cr-ct;
    dy=fr-ft;
    e=(float)sqrt(dx*dx+dy*dy);
    R=e*(celda);
    Rtotal=(float)sqrt(R*R+(mapa[indtx]-mapa[ind])*(mapa[indtx]-mapa[ind]));
```

La variable *dx* y *dy* almacenan la distancia horizontal y vertical, respectivamente, en celdas entre el transmisor y el receptor. En *e* se tiene la distancia, también en número de celdas, de la diagonal que une a ambos elementos. Como se puede observar, *e* es la hipotenusa entre *dx* y *dy*.

R es igual a *e*, pero en vez de estar expresada en número de celdas, lo estará en metros. Para finalizar, *Rtotal* es la distancia que existe entre el punto más alto del transmisor y el del receptor.

A continuación se presenta un ejemplo numérico, el cual no tiene relación alguna con el utilizado anteriormente. En este caso se van a considerar los siguientes datos (**ejemplo explicativo 4**):

- *ft = 2* → *indtx = 607*; *mapa[607] = 721*
- *ct = 5* → *ht = 10*

- *fr= 45* → *ind = 13711*; *mapa[13711] = 98*
- *cr= 166* → *hr = 2*

A partir de estos valores se conocerán los que tendrán *dy* y *e*, los cuales se pueden observar gráficamente en la siguiente figura.

- *dx=166-5=161*
- *dy=45-2=43*

$e = \sqrt{(161)^2 + (43)^2} = 166{,}64$ *celdas es la distancia del TX al Rx sobre el suelo*

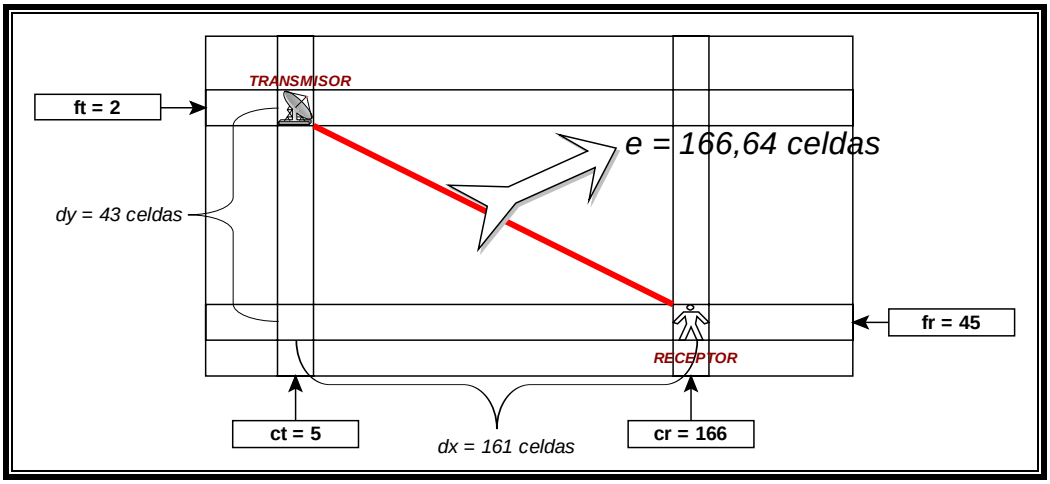


Figura 3.6. Ejemplo explicativo 4. Situación en número de celdas del transmisor y el área de cobertura.

Por último se calculan las distancias *R* y *Rtotal*, las cuales también pueden ser observadas gráficamente a continuación.

- *R=166,64 * 210=34994,4 metros es la distancia del Tx al Rx sobre el suelo.*
- $Rtotal = \sqrt{(34994{,}4)^2 + (721 - 98)^2} = 34999{,}9$ *metros es la distancia en diagonal entre la altura de la antena transmisora y la receptora.*

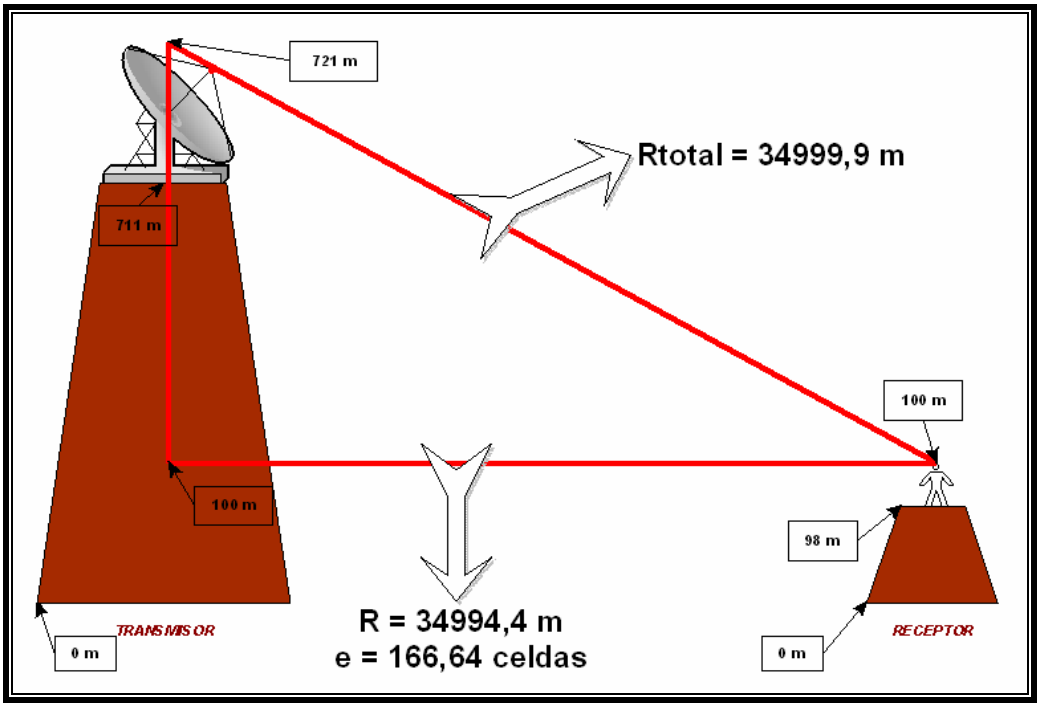


Figura 3.7. Ejemplo explicativo 4. Distancia entre TX y RX.

En el archivo primario, a esta altura del código, se presenta la condición para delimitar si el cálculo de pérdidas se va a realizar sobre un área rectangular o circular. En caso que se seleccione circular (*figura = 1*) y que *R* no sea mayor que el *radio* del área de cobertura (pasado como argumento) a partir del transmisor, se accederá a partir de un *else* a la porción de código presentada a continuación, donde se elevará el perfil para el posterior cálculo de pérdidas. Sin embargo, en el código optimizado, esa comprobación se realiza dentro del fichero "calculaatenuacion.cpp", y más concretamente, al principio del método *CalculaAtenuacion*. Cuando se llegue a ese punto, tendrá lugar una explicación más exhaustiva acerca de la diferencia entre el cálculo de coberturas para las dos figura geométricas mencionadas.

En esta parte del código se realiza una interpolación, con el fin de saber a qué fila y columna del MDT pertenece cada una de las celdas que unen al transmisor con el receptor. Par ello, lo primero es crear dos vectores:

- *x*, donde se almacenarán las columnas,
- *y*, donde lo harán las filas.

El tamaño de estos vectores será igual a la distancia en celdas sobre el suelo entre el transmisor y el receptor (*e*), pero con aproximación al valor entero más alto (*e1*).

```
/*8. Se elimina la condición de devolver -9999 en caso de que figura=1 y R>radio

        /***RESERVA DE MEMORIA***/

        e1=floor(e)+1;
        x=(int*)calloc(e1,sizeof(int));
        y=(int*)calloc(e1,sizeof(int));


        /***CREACIÓN DEL VECTOR X***/

        paso=(e==0) ? 0 : dx/floor(e);
        for (i=0;i<(e1-1);i++){
           aux=ct+i*paso-floor(ct+i*paso);
           if (aux<0.5) x[i]=floor(ct+i*paso);
           else x[i]=ceil(ct+i*paso);
        }

        x[e1-1]=cr;


        /***CREACIÓN DEL VECTOR Y***/

        paso=(e==0) ? 0 : dy/floor(e);
        for (i=0;i<(e1-1);i++){
           aux=ft+i*paso-floor(ft+i*paso);
           if (aux<0.5) y[i]=floor(ft+i*paso);
           else y[i]=ceil(ft+i*paso);
        }

        y[e1-1]=fr;
```

Se procede ahora a la explicación del proceso de creación del vector *x*, tomando de nuevo como refencia el **ejemplo explicativo 4**.

e = 166,64 celdas
dx = 161 celdas
ct = 5 celdas
dy = 43 celdas
ft = 2 celdas

La primera sentencia obliga a hacer *paso* = 0 si se cumple que *e* = 0; es decir, que la celda en que se ubica el receptor sea la misma que la del transmisor; debido al nuevo funcionamiento del código, esto se cumplirá sólo cuando coincida que el transmisor esté en el borde del cuadrante pasado como argumento para el cálculo de pérdidas.

En caso que $e \neq 0$ (situación más frecuente), se calcula la variable *paso* (cociente entre *dx* y el valor entero de *e*) que permitirá la interpolación de las celdas. El fin es conseguir **tranformar el área de cobertura seleccionada en un cuadrado** con lados de tamaño iguales al del vector *e1*, para así poder levantar el perfil a partir de la línea diagonal que una el transmisor con el receptor. En este caso, *paso = 161 / floor (166,64) = 0,969*.

A continuación se presenta cómo serían algunas de las iteraciones de la creción de los vectores *x* e *y*.

- *i = 0* → *¿i < 167-1?* → *Sí*
  *aux = 5 + 0 * 0,969 – floor (5 + 0 * 0,969) = 0*
  ¿0 < 0,5? → *Sí* → *x[0] = floor (5) = 5*

- *i = 1* → *¿i < 167-1?* → *Sí*
  *aux = 5 + 1 * 0,969 – floor (5 + 1 * 0,969) = 0,969*
  ¿0,969 < 0,5? → *No* → *x[1] = ceil (5,969) = 6*

- *i = 2 → ¿i < 167-1? → Sí*
  *aux = 5 + 2 * 0,969 – floor (5 + 2 * 0,969) = 0,938*
  ¿0,938 < 0,5? → No → x[2] = ceil (6,938) = 7

- *i = 16 → ¿i < 167-1? → Sí*
  *aux = 5 + 16 * 0,969 – floor (5 + 16 * 0,969) = 0,504*
  ¿0,504 < 0,5? → No → x[16] = ceil (20,504) = 21

- *i = 17 → ¿i < 167-1? → Sí*
  *aux = 5 + 17 * 0,969 – floor (5 + 17 * 0,969) = 0,473*
  ¿0,473 < 0,5? → Sí → x[17] = floor (21,473) = 21

Como se puede observar, al ser *dx* prácticamente del mismo tamaño que *e1*, hasta la iteración número 18 no se vuelve a utilizar una misma columna para dos posiciones consecutivas del vector *x*.

- *i = 18 → ¿i < 167-1? → Sí*
  *aux = 5 + 18 * 0,969 – floor (5 + 18 * 0,969) = 0,442*
  ¿0,442 < 0,5? → Sí → x[18] = floor (22,442) = 22

- *i = 164 → ¿i < 167-1? → Sí*
  *aux = 5 + 164 * 0,969 – floor (5 + 164 * 0,969) = 0,916*
  ¿0,916 < 0,5? → No → x[164] = ceil (163,916) = 164

- *i = 165 → ¿i < 167-1? → Sí*
  *aux = 5 + 165 * 0,969 – floor (5 + 165 * 0,969) = 0,885*
  ¿0,885 < 0,5? → No → x[165] = ceil (164,885) = 165

- *i = 166 → ¿i < 167-1? → No → Se sale del bucle for.*

El último valor del vector *x* será la columna en la que se encuentre el receptor, como se puede deducir de la sentencia final con que concluye la construcción de dicho vector:

- *x [e1-1] = cr* → *x[166] = 166.*

Nota: si *paso* fuese igual a 0 (transmisor y receptor en la misma celda), no se cumpliría la condición del *for* para la iteración *i* = 0, por tanto *x [1-1] = x [0] = cr = 5*; es decir, en el vector *x* estaría almacenada la columna del receptor (y transmisor en este caso), y en *y* la fila del mismo.

Para la creación de las filas, *paso = 43 / floor (166,64) = 0,259*. El funcionamiento para la creación del vector *y* es similar que para *x*. A continuación se muestran algunos de los valores para unas determinadas iteraciones:

i	y [i]
0	2
1	2
2	3
3	3
4	3
5	3
6	4
164	44
165	44
166	44

Tabla 3.5. Ejemplo explicativo 4. Interpolación del vector *y*.

Al ser *dy* bastante más pequeño que *e1*, la interpolación es más significativa que para *x*.

Para terminar con este bloque, se muestra el significado gráfico de la interpolación.

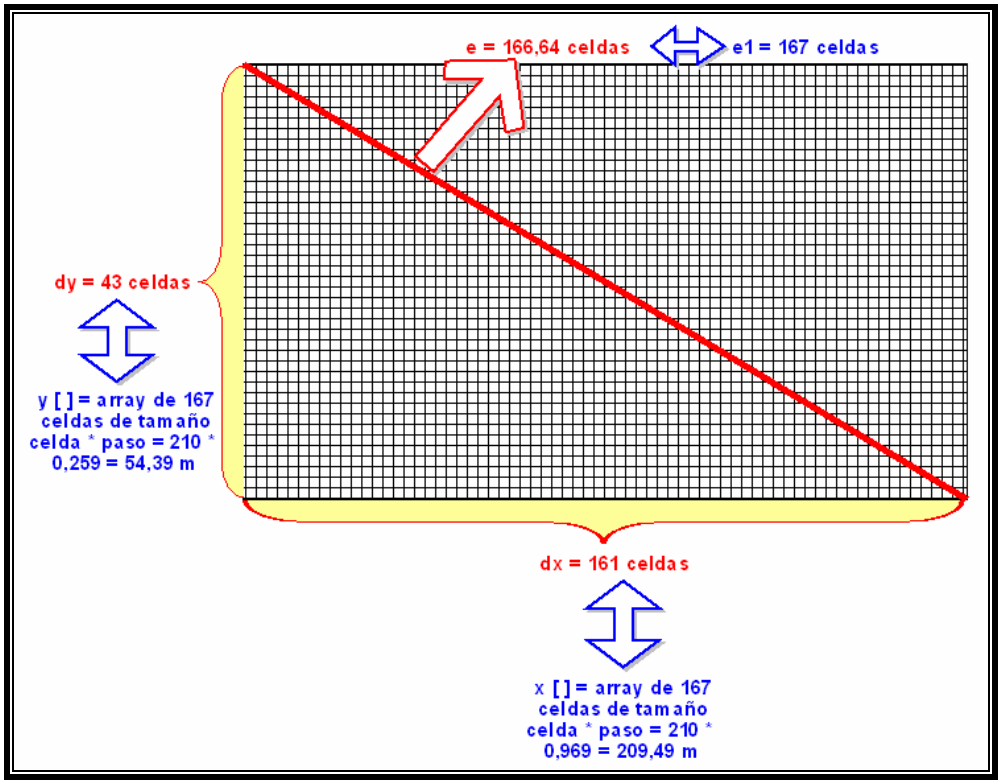


Figura 3.8. Ejemplo explicativo 4. Interpolación de las celdas entre el TX y el RX.

Ahora llega el momento en el que **se procede a crear definitivamente el vector perfil**, junto a otros también de suma importancia.

El primer paso es reservar memoria con la sentencia *calloc* para los nuevos vectores que se van a formar. En el código primario se reservaba para los vectores *perfil_flecha* e *indices_ptoscorte*, pero aquí no procede porque estos dos vectores serán utilizados en el método *CalculaAtenuacion*. Además, se crea un nuevo vector (*ind_calc*), donde se almacenarán los índices de las celdas que van a conformar el presente perfil desde la celda perteneciente al borde del cuadrante de cobertura, hasta el transmisor.

Seguidamente, un bucle *for* con tantas iteraciones como celdas forman el perfil, va a servir para rellenar el vector *perfil* con las alturas de cada una de las celdas, y el anteriormente comentado vector *ind_calc*.

```
        /***SE HALLA EL PERFIL***/

        perfil=(float*)calloc(e1,sizeof(float));
/*9.  Se  suprime  la  reserva  de  memoria  para  los  vectores  'perfil_flecha'  e
'indices_ptoscorte'*/
        vector_distancias=(float*)calloc(e1,sizeof(float));
        ind_calc=(int*)calloc(e1,sizeof(int));
/*10.  En  'ind_calc'  se  guardan  los  índices  que  componen  el  perfil  completo  de  la
iteración actual (desde el transistor a cada una de las celdas que se encuentre en el
exterior del área de cobertura)*/

        if (!perfil){
           printf("No queda memoria disponible para perfil\n");
           exit(1);
        }

        for(j=0;j<e1;j++){
           ind=(long) ncols1*y[j]+x[j];
           perfil[j]=mapa[ind];
           ind_calc[j]=ind;
/*11. Se rellena el array 'ind_calc' como se ha comentado anteriormente*/
        }

        perfil[e1-1]=perfil[e1-1]-hr;
/*12.  Se  resta  la  altura  del  receptor  a  'perfil[]',  para  ir  sumándola  y  restándola
sucesivamente en cada iteración dentro del método CalculaAtenuacion*/
```

Con el **ejemplo numérico 4** se mostrará el funcionamiento de dicho bucle en algunas de las iteraciones.

- $j = 0 \rightarrow$ ¿$j < 167$? $\rightarrow$ Sí

  $ind = 301 * y[0] + x[0] = 301 * 2 + 5 = 607$

  $perfil[0] = mapa[607] = 721$

  $ind\_calc[0] = 607$

- $j = 1 \rightarrow$ ¿$j < 167$? $\rightarrow$ Sí

  $ind = 301 * y[1] + x[1] = 301 * 2 + 6 = 608$

  $perfil[1] = mapa[608] = 702$

  $ind\_calc[1] = 608$

- *j = 2 → ¿j < 167? → Sí*
  *ind = 301 * y [2] + x [2] = 301 * 3 + 7 = 910*
  perfil [2] = mapa [910] = 508
  ind_calc [2] = 910

- *j = 3 → ¿j < 167? → Sí*
  *ind = 301 * y [3] + x [3] = 301 * 3 + 8 = 911*
  perfil [3] = mapa [911] = 487
  ind_calc [3] = 911

- *j = 164 → ¿j < 167? → Sí*
  *ind = 301 * y [164] + x [164] = 301 * 44 + 164 = 13408*
  perfil [164] = mapa [13408] = 89
  ind_calc [164] = 13408

- *j = 165 → ¿j < 167? → Sí*
  *ind = 301 * y [165] + x [165] = 301 * 45 + 165 = 13710*
  perfil [165] = mapa [13710] = 98
  ind_calc [165] = 13710

- *j = 166 → ¿j < 167? → Sí*
  *ind = 301 * y [166] + x [166] = 301 * 45 + 166 = 13711*
  perfil [166] = mapa [13711] = 100
  ind_calc [166] = 13711

- *j = 167 → ¿j < 167? → No → Se sale del bucle for.*

Tras la salida del bucle, se procede a restar la altura física del receptor a la que se encuentra almacenada dentro de *perfil* en la posición de éste (en esta ocasión será *perfil [166] = 100 – 2 = 98*). Este paso es necesario, ya que será en el método *CalculaAtenuacion* donde se sumará y restará *hr* según el receptor sea desplazado

posición tras posición dentro del vector *perfil* generado, desde la celda del actual receptor, hasta al transmisor.

El siguiente bloque de código corresponde a la creación del vector de nombre *vector_distancias*. Es necesaria realizar una interpolación, puesto que la distancia entre celdas es en metros el valor de *cellsize* (en este caso 210 metros), pero esto se cumple para aquellas que se encuentran en la misma fila o columna del transmisor. Así pues, cuando el perfil está formado por celdas dispuestas sobre el MDT en diagonal, esa distancia entre celdas se modifica, y se calculará como *R / floor (e)*; excepto para el conocido caso en que *e = 0* (transmisor y receptor en la misma celda), donde la distancia entre celdas será claramente 0 metros.

Se vuelve a mostrar el funcionamiento del código de forma numérica para su mayor comprensión (**ejemplo explicativo 4**).

```
         /***CREACIÓN DEL VECTOR DE DISTANCIAS***/

         paso2=(e==0) ? 0 : R/floor(e);
         for (i = 0; i < (e1-1); i++) {
            vector_distancias[i]=i*paso2;
         }

         vector_distancias[e1-1]=R;

         if(ind!=indtx)
         mapa[ind]=mapa[ind]-hr;
/*13. Se resta la altura 'hr' a 'mapa[ind]' para sucesivos levantamientos de perfil*/
```

Como *e = 164,64 ≠ 0 → paso2 = 34994,4 / floor (166,64) = 210,8*; es decir, la distancia entre dos celdas contiguas del perfil será de 210,8 metros (y no 210 metros). Con este dato ya se puede proceder a rellenar el vector *vector_distancias*.

- *i = 0 → ¿i < 167 - 1 ? → Sí*
  *vector_distancias [0] = 0 * 210,8 = 0*

- *i = 1 → ¿i < 167 - 1 ? → Sí*
  *vector_distancias [1] = 1 * 210,8 = 210,8*

- *i = 2 → ¿i < 167 - 1 ? → Sí*
  *vector_distancias [2] = 2 * 210,8 = 421,6*

- *i = 3 → ¿i < 167 - 1 ? → Sí*
  *vector_distancias [3] = 3 * 210,8 = 632,4*

- *i = 164 → ¿i < 167 - 1 ? → Sí*
  *vector_distancias [164] = 164 * 210,8 = 34571,2*

- *i = 165 → ¿i < 167 - 1 ? → Sí*
  *vector_distancias [165] = 165 * 210,8 = 34782*

- *i = 166 → ¿i < 167 - 1? → No → Se sale del bucle for.*

El bucle ha completado los valores del *vector_distancias* desde la posición 0 a la 165. La última posición de éste (166) se adjudica al valor *R*; es decir, la distancia que existe sobre el suelo desde el transmisor hasta el receptor. Concretamente para este caso *vector_distancias [166] = R = 34994,4*.

En el siguiente bloque se realiza la llamada al método *CalculaAtenuacion*. Difiere muy significativamente respecto al programa primario, puesto que en el original no aparece un bucle *for* como el que se va a presentar para establecer dicha llamada. **Este bucle es una pieza fundamental** para conseguir aumentar la velocidad de cálculo del programa.

```
G=0;

if (e1==1) {
   atenuacion=-9999;
   campo[ind]=atenuacion;
}
```

```
        else{
/***14. BUCLE PARA REALIZAR LA LLAMADA AL MÉTODO CALCULAATENUACION PARA AQUELLOS INDICES
QUE SE DEBEN CALCULAR DENTRO DEL PERFIL COMPLETO DE ESTA ITERACIÓN***/
            ep=e1-1;
            for(i=(e1-1);i>0;i--){

               if(campo[ind_calc[ep]]==(-9999)){
                  atenuacion=CalculaAtenuacion(perfil,celda,frecuencia,k,
                  vector_distancias,ep,hr,R,Rtotal,e1,ft,ct,y,x);

                  campo[ind_calc[ep]]=atenuacion - G;
               }

               ep=ep-1;
            }

         }

         free(x);
/*15. Se libera el espacio de memoria de los vectores 'x' e 'y' para cada iteración al
igual que el vector 'perfil', 'vector_distancias' e 'ind_calc'*/
         free(y);
         free (perfil);
         free (vector_distancias);
/*16. De nuevo se elimina la liberación de memoria de los vectores 'indices_ptoscorte' y
'perfil_flecha'*/
         free (ind_calc);
/*17. Se liberan los índices de este perfil para volver a rellenarlo en la siguiente
iteración*/
/*18. Se elimina la llave que cierra el bucle else abierto para el cálculo de pérdidas
en caso de que no se cumpla la condición ya suprimida en el comentario número 8*/
   } /***FIN DE LA CONDICIÓN DE CELDA EN LA QUE SE HAN CALCULADO LAS PÉRDIDAS***/
```

En la primeras sentencia se asigna como ganancia 0dB´s. Posteriormente se fuerza en caso de ser *e1 = 1* (receptor en la misma celda que el transmisor), a que el valor del vector *campo* (donde se almacenan los resultados de las pérdidas) sea igual a *-9999*; o lo que es lo mismo, tenga valor nulo y no se proceda a realizar el cálculo de pérdidas.

En caso contrario (el perfil está formado por más de una celda), se procede al cálculo de la cobertura. Para ello se entra al bucle *for*, pero antes, se asigna a ***ep* el valor *e1 – 1*, ya que dicha variable será la que funcione como la posición de todas las alturas dentro de *perfil*, y de las distancias dentro de *vector_distancias***. Así pues,

para el ejemplo anterior, donde *e1 = 167* (el perfil está formado por 167 celdas), los vectores *perfil* y *vector_distancias* tendrán almacenados 167 valores desde la posición 0 hasta la 166 (*perfil [0 ... 166]* y *vector_distancias [0 ... 166]*).

Una vez dentro del bucle *for*, se iterará celda a celda desde la que se encuentre en el extremo del área de cobertura (valor último del vector *ind_calc*), hasta la celda contigua a la del transmisor.

Por ejemplo, el primer valor de *i* será *166*, y se irá restando una unidad mientras que *i* sea mayor que *0*, por lo que la última iteración será para *i = 1*. Resumiendo, se iterará desde *ind_calc [166]* (extremo del cuadrante de cobertura) hasta *ind_calc [1]* (celda contigua a la del transmisor), acercándose en cada iteración una celda más hacia el transmisor (*ind_calc [0]*).

Para cada una de las iteraciones, se comprueba si el valor de *campo* para el índice actual es igual a *-9999*; en caso afirmativo se procede a llamar al método *CalculaAtenuacion* para calcular las pérdidas. ¿Por qué se realiza esta comprobación? Muy sencillo, anteriormente se han rellenado las posiciones del vector *campo* (donde como ya se ha comentado, se almacenrán los resultados de las pérdidas para el área de cobertura seleccionada) con el valor *-9999*. Cuando se calculen las pérdidas para un determinado índice, quedarán almacenadas en dB`s en el vector *campo*. Posteriormente, cuando se vuelva a levantar otro perfil, puede darse el caso que la misma celda en la que anteriormente se calcularon las pérdidas, forme parte de este nuevo perfil; así, cuando se entre al bucle *for* con este perfil, no se procederá a calcular las pérdidas para este índice debido a que las pérdidas almacenadas en *campo* para él son diferentes de *-9999*.

Por cierto,antes de pasar a la sigiente iteración, a la vez que la variable *i* disminuya en una unidad, también lo hará *ep*.

Observese las tres primeras iteraciones una vez levantado el perfil del **ejemplo explicativo 4**. Como $e1 \neq 1$, se ejecuta el *else*.

- *ep = 167 -1 =166*

  i = 166 → ¿i > 0? → Sí

  ¿campo [ind_calc [ep]] = campo [ind_calc [166]] = campo [13711] = -9999? → Sí (se supone que para el ejemplo aun no se han calculado las pérdidas para este índice).

  Llamada al método CalculaAtenuacion (devolverá el valor de las pérdidas, por ejemplo, atenuación = 78).

  campo [13711] = 78 – 0 = 78 (las unidades de este valor son dB´s)

- *ep = 166 -1 =165*

  i = 165 → ¿i > 0? → Sí

  ¿campo [ind_calc [ep]] = campo [ind_calc [165]] = campo [13710] = -9999? → No (se supone que para el ejemplo ya se ha calculado las pérdidas para este índice en otro perfil anterior al actual: por ejemplo podría darse el caso que campo [13710] = 75)

- *ep = 165 -1 =164*

  i = 164 → ¿i > 0? → Sí

  ¿campo [ind_calc [ep]] = campo [ind_calc [164]] = campo [13408] = -9999? → Sí (se supone que para el ejemplo aun no se ha calculado las pérdidas para este índice)

  Llamada al método CalculaAtenuacion (devolverá el valor de las pérdidas, por ejemplo, atenuación = 74).

  campo [13408] = 74 – 0 = 74

De esta manera se consigue aumentar la velocidad aun más, ya que al hecho de levantar el perfil en un número mucho más reducido de ocasiones que para el código primario, se le une que solamente se llamará al método *CalculaAtenuacion* una vez por cada celda a calcular.

En el código primario se levanta un perfil para cada una de las celdas en las que se quiera calcular las pérdidas, y con dicho perfil se procede a llamar al

método *CalculaAtenuacion*. Para el código del que ahora se dispone, se crean los perfiles únicamente en las celdas que forman parte de los extremos del área de cobertura a calcular, y con cada perfil disponible se aprovecha para llamar al método *CalculaAtenuacion* tantas veces como celdas que compongan el perfil, y en las que todavía no se hayan calculado los pérdidas, existan. De esta manera se consigue aumentar la velocidad para el cálculo de pérdidas.

El funcionamiento dentro del método *CalculaAtenuacion* se explicará en el siguiente punto de la memoria. Por ahora basta con saber que dicho método devolverá las pérdidas en dB´s a partir de la variable *atenuación*, y ésta será almacenada (tras restarle la ganancia *G*) en la posición de *campo* correspondiente al índice para el que se han calculado las pérdidas.

Una vez calculadas todas las pérdidas necesarias en el perfil levantado, se procede a liberar el espacio de memoria correspondiente a los vextores *x*, *y*, *perfil*, *vector_distancias* e *ind_calc*; los cuales volverán a ser creados y utilizados en la siguiente iteración en la que se levante otro perfil.

La nueva porción de código que acontece tampoco existe en el código primario. Se asigna el valor nulo *-9999* a todas aquellas celdas que no cumplan la condición explicada anteriormente de forma detallada, por la cual se sabe si el índice del receptor de la actual iteración forma parte del borde del cuadrante de cobertura (es decir, a este *else if* se entraría en caso de no haber levantado perfil ni calculado pérdidas), y que además, se encuentren fuera del área de cobertura a calcular.

Con esta acción, una vez recorrido todo el MDT, todas aquellas celdas que se encuentren fuera del área de cobertura calculada, tendrán valor nulo *-9999*. Cuando la herramienta visual de *RADIOGIS* lea este valor, no lo representará sobre el grid en pantalla.

```
   else if ((fr<fa) || (fr>fa+nfils-1) || (cr<ci) || (cr>ci+ncols-1)){
/*19. Con esta condición, se asigna valor 'campo[ind]' a todas las celdas que estén
fuera del área de cobertura*/
```

```
      campo[ind]=-9999;
   }

   if(ind!=indtx) mapa[ind]=mapa[ind]-hr;
/*20. Se resta la altura del receptor a todas aquellas celdas que no pertenezcan al área
de cobertura seleccionada*/
} /***FIN DEL BUCLE PRINCIPAL***/
```

Por último, al finalizar una iteración (se haya levantado o no perfil) se resta la altura del receptor (siempre que no sea la misma celda del transmisor) al vector *mapa* en la posición del índice con el que se está trabajando (*ind*), para que dicha *hr* no influya cuando se levante un nuevo perfil perteneciente a una próxima iteración.

Con estas líneas finaliza el bucle principal, de tal forma, cuando se haya completado la última iteración (*fr = nrows1 - 1* y *cr = ncols1 – 1)*, ya se habrá finalizado el cálculo de la cobertura seleccionada, y se procederá a sacar los resultados en pantalla.

La ejecución del cálculo finaliza con la escritura de las pérdidas en el archivo de texto *resultado.txts*. En los índices pertenecientes al área de cobertura se guardará un número entero correspondiente al valor de las pérdida en dB's (las que se han almacenado previamente en el vector *campo*); por otra parte, los que estén fuera de dicha delimitación, tendrán almacenados como resultado el valor *-9999*, lo cual se tomará en el entorno de *RADIOGIS* como un valor nulo que no repercutirá en la representación de resultados sobre el grid.

```
/***ESCRITURA DEL FICHERO DE CAMPO EN DISCO***/

if ((resultados=fopen(argv[3],"a"))!=NULL){

   filasar=((yllcorner2+cellsize2*nrows2)-(yllcorner1+cellsize1*nrows1))/
   cellsize2;
   filasab=nrows2-nrows1-filasar;
   coli=(xllcorner1-xllcorner2)/cellsize2;
   cold=ncols2-ncols1-coli;

      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
```

```
            fprintf(resultados," ");
         }

         fprintf(resultados,"\n");
      }

      for (i=0;i<nrows1;i++){
         for (y1=0;y1<coli;y1++){
            fprintf(resultados,"-9999");
            fprintf(resultados," ");
         }

         for (j=0;j<ncols1;j++){
            ind1=(long) ncols1*i+j;
            escribe_float(resultados,&(campo[ind1]));
            fprintf(resultados," ");
         }

         for (y1=0;y1<cold;y1++){
            fprintf(resultados,"-9999");
            fprintf(resultados," ");
         }

         fprintf(resultados,"\n");
      }

      for (x1=0;x1<filasab;x1++){
         for (y1=0;y1<ncols2;y1++){
            fprintf(resultados,"-9999");
            fprintf(resultados," ");
         }

         fprintf(resultados,"\n");
      }

      fclose (resultados);
      printf("El campo está  en disco\n");
}

else{
      printf("No se puede abrir fichero de salida\n");
      exit (1);
}

free(campo);
free(mapa);

comprueba=fopen(argv[22],"w");
fprintf(comprueba,"\n");
```

```
fclose (comprueba);
end = clock();

if ((tiempo=fopen("c:\\arcgis\\RAGIS\\tiempo.txt","w"))!=NULL){
   fprintf(tiempo,"The time was: %f\n", (end - start) / CLK_TCK);
   fclose(tiempo);
}

} /***FIN DE MAIN***/
```

Por último, se creará un archivo llamado *tiempo.txt*, en el cual se reflejará el tiempo de cálculo de la cobertura seleccionada. Gracias a este dato, se puede comparar la diferencia entre los tiempos de cálculo del programa primario y el optimizado.

3.4.2. Modificación en el fichero calculaatenuacion.cpp.

Sobre las modificaciones realizadas en este fichero, **se va a centrar la atención principalmente en la primera parte del método *CalculaAtenuación***, que es al que se llama desde el método *main* del fichero "principal.cpp", una vez que se desea proceder al cálculo de pérdidas por el método UITR-526. En el método *CalculaAtenuación* se realizan llamadas, que sirven como apoyo al cálculo, a otros cuatro métodos (*CalculaPerfil_flecha*, *LOS*, *CalculaFuniculo* y *espacio libre*) que forman parte del mismo fichero "calculaatenuacion.cpp".

Siguiendo con el **ejemplo explicativo 4**, en el momento en que se realiza la llamada al método *CalculaAtenuacion*, los valores calculados en el método *main*, y que se pasan como argumentos serían los siguientes:

```
/***5.- MÉTODO CALCULAATENUACIÓN***/

float CalculaAtenuacion(float *perfil, float cellsize, float frec, float K, float
*vector_distancias, int ep, float hr, float R, float Rt, int e1, int ft, int ct, int *y,
int *x) {
/*21. Se pasa como argumento 'ep' en vez de npuntos, y también 'hr', puesto que ahora el
sumar y restar la altura de la antena receptora no se hace en principal.cpp, sino aquí;
además se elimina 'perfil_flecha' e 'indices_ptos corte'*/
```

- **perfil = [721, 702, 508, 487, …, 89, 98, 98]**
- **cellsize = 210 metros**
- **frec = 900 MHz**
- **K = 4 / 3**
- **vector_distancias = [0, 210.8, 421.6, 632.4, …, 34571.2, 34782, 34994.4]**
- **ep = 166 (su valor irá decrementando en cada iteración)**
- **hr = 2 metros**
- **R = 34994,4 metros**
- **Rt = 34999,9 metros**
- **e1 = 167 celdas**
- **ft = 2 (número de celda)**
- **ct = 5 (número de celda)**
- **y = [2, 2, 3, 3, ..., 44, 44, 44]**
- **x = [5, 6, 7, 8, …, 164, 165, 166]**

Como se ha explicado con anterioridad, en el programa optimizado con un único levantamiento del perfil en el método *main*, se llamará a *CalculaAtenuación* en un número elevado de ocasiones; sin embargo, en el primario, se realiza una única llamada por cada perfil levantado.

Así pues, tanto el código primario como el optimizado, calcularán las pérdidas de forma similar, pero el primero lo hará a partir de los vectores *perfil* y *vector_distancias*; y el segundo de *perfil_actual* y *vector_distancias_actual*.

De tal manera, en el optimizado, con un único levantamiento del perfil en el método *main*, se llamará a *CalculaAtenuación* en un número elevado de ocasiones; sin embargo, en el primario se realiza una única llamada por cada perfil levantado.

El bloque que a continuación se presenta está formado por la definición de las variables creadas en el método.

En el código primario, a diferencia del optimizado, se declaran y completan los vectores *perfil_flecha* e *indices_ptoscorte* en el método *main*. La razón por la que en la optimización no se realiza así, es porque se irán modificando con cada llamada a método.

La modificación más importante está en la inclusión de cuatro nuevos vectores: *perfil_actual*, *vector_distancias_actual*, *dist_actual* y *paso_actual*. El significado de estos vectores se explicará en los siguientes bloques, en los cuales tomarán sus valores correspondientes.

```
   float *flecha;
   float Ltotal=0, LD=0, L0=0, Llluvia=0, Lgases=0, Ldi=0, sumatorioLD=0;
   int vision;
   float roo = 7.5;
   float tasa = 25;
   char polarizacion = 'h';
   int i, j;
   float *alturas_obs;
   float *distancias_obs;
   int num_vanos;
   float d1, d2, vi, hi, tang, vimax;
   float sumatorioLsdi=0, Lsdi=0, Lsd_max=0;
   double Cn=0;
   double a, b, c;
   float htx, hrx, posicion_tx, posicion_rx;
   int long_ptos,l;
   float *perfil_flecha;
/*22.  Además  es  aquí  donde  se  declaran  los  vectores  'perfil_flecha'  e
'indices_ptoscorte', los que también se irán modificando con cada llamada a método*/
   int *indices_ptoscorte;
   float *perfil_actual;
/*23. Se declara el nuevo vector 'perfil_actual' a reutilizar en cada llamada a método*/
   float *vector_distancias_actual;
/*24. Se declara el nuevo vector 'vector_distancias_actual' a reutilizar en cada llamada
a método*/
   float dist_actual;
/*25. Se declara 'dist_actual' y 'paso_actual' para crear el 'vector_distancias_actual'
adecuado cuando no se calculen las pérdidas en celdas que no estén en el exterior del
área de cobertura*/
   float paso_actual;
```

La siguiente porción de código no es más que la reserva de memoria de cada uno de los vectores definidos anteriormente.Además, si $K = 0$, saldrá de la ejecución del código. Pero esa situación no suele ocurrir, puesto que K es un valor introducido en *RADIOGIS* como argumento (*k*), y no es más que una constante de valor *4 / 3*.

Pero antes de esto, se define e inicializa *ep1*. Esta variable es similar a *e1* (distancia en celdas sobre el suelo entre el transmisor y el receptor, pero con aproximación al valor entero más alto para cada una de las iteraciones), aunque sólo tendrán el mismo valor en la primera de las iteraciones de la llamada. El valor de *ep1* irá decrementando en función del valor de *ep*, el cual, como se comentó en el apartado 1.4.1, variará desde (*ep1 – 1*) hasta *1*; así pues, los valores que puede tomar *ep1* irán desde *e1* hasta *2* (se evita la última iteración, por la cual *ep = 0*, y por tanto *ep1 = 1*, o lo que es lo mismo, la celda de dicha iteración sería la del transmisor, y en ésta, como ya es sabido, no se desea calcular las pérdidas).

```
   int ep1;
/*26. Se declara 'ep1'*/

   ep1=ep+1;
/*27. 'ep1' es lo mismo que el 'e1', pero para cada una de las iteraciones de 'ep'*/

   flecha = (float*)calloc(ep1,sizeof(float));
   alturas_obs = (float*)calloc(ep1,sizeof(float));
   distancias_obs = (float*)calloc(ep1,sizeof(float));
   perfil_actual = (float*)calloc(ep1,sizeof(float));
/*28. Se reserva memoria para los nuevos vectores*/
   vector_distancias_actual = (float*)calloc(ep1,sizeof(float));
   perfil_flecha = (float*)calloc(ep1,sizeof(float));
   indices_ptoscorte = (int*)calloc(ep1,sizeof(float));

   if (K == (float)0){
      return -1;
   }
```

En los siguientes bloques se completan las variables mencionadas. Se va a explicar el funcionamiento del código a partir del anteriormente mencionado **ejemplo explicativo 4**.

La primera de las próximas porciones de código se ejecutará cuando tenga lugar la primera llamada al método una vez creado en *main* el vector *perfil* y *vector_distancias*, entre otros. En estas circunstancias siempre se cumplirá que *ep1 = e1*, es decir, la celda en que se van a calcular las pérdidas es una de las que conforman el borde del área de cobertura pasada como parámetro para el cálculo, o lo que es lo mismo, de todas las celdas que componen el perfil, ésta es la más alejada del transmisor.

Nota: esta afirmación será correcta siempre que se cumpla que el transmisor esté ubicado dentro del cuadrante de cobertura. Si no fuese así y estuviese en una celda perteneciente al borde de dicha área (o cercano a éste), no tiene por qué cumplirse que *ep1 = e1*, ya que aunque ésta celda A sea la más lejana en el *vector_distancias*, podría haber formado parte de otro perfil levantado anteriormente desde una celda B que se encuentre en la misma fila o columna del borde del cuadrante en que se están el transmisor y la celda A.

```
   if (ep1==e1){
/*29. Creación de los vectores 'perfil_actual' y 'vector_distancias_actual' para las
celdas exteriores del área de cobertura a calcular*/
      for (i=0; i<ep1; i++){
         perfil_actual[i]=perfil[i];
         vector_distancias_actual[i]=vector_distancias[i];
      }

      perfil_actual[ep]=perfil_actual[ep]+hr;
   }
```

En este caso, los vectores *perfil_actual* y *vector_distancias_actual* serán exactamente iguales que *perfil* y *vector_distancias* respectivamente. La única salvedad será sumar a la última de las alturas de *perfil_actual* (*perfil_actual[ep]*) la del receptor (*hr*), puesto que en esa celda es donde se va a proceder al cálculo de las pérdidas en la señal recibida por el receptor.

De esta manera, para el ejemplo numérico, se parte de los siguientes valores:

- **perfil = [721, 702, 508, 487, …, 89, 98, 98]**

- **vector_distancias = [0, 210.8, 421.6, 632.4, …, 34571.2, 34782, 34994.4]**
- **hr = 2 metros**
- **ep = 166**
- **ep1 = 167 celdas**

Los vectores obtenidos son:

- *pefil_actual = [721, 702, 508, 487, …, 89, 98, **100**]*
- *vector_distancias_actual = [0, 210.8, 421.6, 632.4, …, 34571.2, 34782, **34994.4**]*

A partir de aquí se procede a realizar el cálculo de las pérdidas del radioenlace de la misma manera que en el código primario.

El código mostrado se activará el mismo número de veces que se realice un levantamiento de perfil en el método *main* (excepto si por casualidad, como ya se ha explicado, el transmisor se encontrase en el borde del área de cobertura a calcular). Esta afirmación es exacta siempre que se desee calcular un área rectangular, si por el contrario el cálculo fuera para un círculo, esto no sería así, pero ese caso será explicado en breve más adelante.

Sin embargo, como se observa seguidamente, en la mayoría de ocasiones, las celdas no pertenecen al borde del área de cobertura, y habrá que realizar un trabajo previo más complejo antes de proceder al cálculo. El problema se centra en el *vector_distancias_actual*, *R* y *Rt*.

Volviendo al ejemplo, supongamos que en el método *main*, en la segunda iteración para el actual perfil (*ep = 165*), resulta que ya se han obtenido sus pérdidas en el proceso de cálculo de otro perfil previo al actual. De tal manera no se produciría la llamada al método *CalculaAtenuacion* para este caso.

Supongamos que en la siguiente iteración en *main*, para esa celda aun no se conocen las pérdidas y es necesario calcularlas. Esto significa que se entra en el método *CalculaAtenuacion* con *ep = 164*, que conlleva a que *ep1 = 165*.

```
   else{
/*30. Creación de los vectores 'perfil_actual' y 'vector_distancias_actual' para las
celdas interiores del área de cobertura a calcular*/
      dist_actual=sqrt((pow((y[ep]-ft)*cellsize,2))+(pow((x[ep]-ct)*cellsize,2)));
      paso_actual=dist_actual/ep;
      for (i=0; i<ep1; i++){
         perfil_actual[i]=perfil[i];
         vector_distancias_actual[i]=i*paso_actual;
      }

      perfil_actual[ep]=perfil_actual[ep]+hr;
      R=vector_distancias_actual[ep];
/*31. Además del vector de las distancias, hay que modificar 'R' y 'Rt'*/
      Rt=(float)sqrt(R*R+(perfil_actual[0]-perfil_actual[ep])*(perfil_actual[0]
      -perfil_actual[ep]));
   }
```

El vector *perfil_actual* se creará de la misma forma que anteriormente, es decir, recorriendo *perfil* desde su posición *0* hasta *ep1-1 = ep = 164*, y sumándo de nuevo la altura del receptor en esa última posición. Con todo ello, el vector obtenido es *pefil_actual = [721, 702, 508, 487, ..., **91**]*.

El problema se da para *vector_distancias_actual* puesto que no basta con recorrer el *vector_distancias* pasado como argumento y almacenar en el actual hasta el índice en que se encuentre el receptor; en este caso, hay que calcular de nuevo el *vector_distancias_actual* de la misma forma en que en el *main* se completaba el *vector_distancias*.

La causa por la que se produce este hecho se puede observar a partir de la figura siguiente:

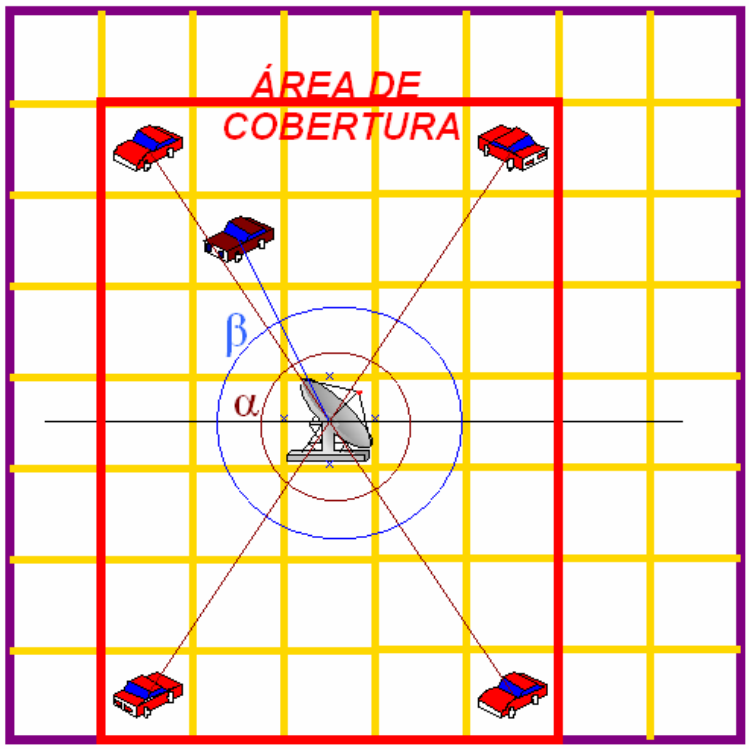


Figura 3.9. Diferentes ángulos entre TX y RX = diferentes distancias entre celdas.

Los radioenlaces formados por los cuatro vehículos rojos con el transmisor, forman un ángulo idéntico de αº con respecto al eje de abcisas. Todos los receptores que se encuentren en este ángulo (independientemente de la distancia que les separe del transmisor) tendrán similar *vector_distancias* (para similares índices, claro está) debido a que la distancia entre celdas será la misma. Sin embargo, en el radioenlace del receptor marrón el ángulo es de βº, por lo que su *vector_distancias* será totalmente diferente a los anteriores al ser también diferente la distancia entre celdas.

El funcionamiento del programa conlleva a que, excepto para el receptor que forme parte del borde del cuadrante de cobertura a calcular (el más alejado), se recalcule la distancia y se cree el nuevo *vector_distancias_actual* en cada caso, independientemente de que el ángulo del radioenlace pueda volver a ser igual del formado por el receptor más alejado (αº en el caso del dibujo).

Véase el ejemplo numérico, teniendo en cuenta que a partir de los argumentos pasados, se obtiene: *y[ep] = y [164] = 44,* y *x[ep] = x [164] = 164*

Como se acaba de explicar, la distancia sobre el suelo entre el centro de la celda del transmisor y la del receptor (*dist_actual*, que en el método *main* equivale a la variable *R*) es diferente que en el primer caso (*ep = 166*), puesto que ahora, la distancia sobre el suelo entre dos celdas contiguas (*paso_actual*) ha variado.

- $dist\_actual = \sqrt{((44-2)*210)^2 + ((164-5)*210)^2} = 34535.26459$ metros.
- $paso\_actual = 34535.26459/164 = 210.5808816$ metros.

En consecuencia el *vector_distancias_actual* es ligeramente diferente al *vector_distancias*: *vector_distancias_actual = [0, 210.5, 421.1, 631.7, ..., **34535,2**]*.

Además, una vez creados el perfil y el vector de distancias, a lo largo del cálculo de pérdidas por el método UITR-526, se utilizan otros datos como es *R* y *Rt*. En el caso de la primera iteración, se sigue usando las distancias calculadas en el método *main*, pero en las posteriores este valor varía y debe ser recalculado. El motivo es el mismo que obliga a modificar el *vector_distancias_actual*. *R* equivale a la última posición del vector mencionado, o lo que es lo mismo, *dist_actual*.

- *R = vector_distancias_actual [164] = 34535,2* metros.
- $Rtotal = \sqrt{(34535,2)^2 + (721-91)^2} = 34540,9$ metros.

Para finalizar se presenta el <u>último bloque</u> creado para conseguir la optimización. En él, se abre un bucle *if*, el cual propone como condición para activarse que *figura = 1*, y *R > radio*. Este bucle se encontraba en el método *main* en el programa primario.

```
   if ((figura==1) && (R > radio)){
/*32. Bucle if con el que se devuelve -9999 y se libera la memoria reservada con calloc
al principio de la llamada, en caso que se quiera calcular un área de cobertura
circular, y dicha altura del MDT no pertenezca a esa circunferencia de cálculo*/
      free(distancias_obs);
      free(alturas_obs);
      free(flecha);
```

```
        free(perfil_actual);
        free(vector_distancias_actual);
        free(perfil_flecha);
        free(indices_ptoscorte);

        return -9999;

    }
```

La variable *figura* se pasa como argumento desde el entorno visual de *RADIOGIS* (*ArcMap*); si se decide calcular las pérdidas en un rectángulo su valor será *0*, y si por el contrario es en un círculo, *1*.En caso que *figura = 1*, el funcionamiento será idéntico al del área rectangular, la única diferencia se encuentra en este bucle. Los argumentos introducidos que definirán el área circular serán las coordenadas del transmisor (*ct* y *ft*), y el radio de la circunferncia (*radio*) a partir de dicho transmisor (éste se encontrará en el centro del área a calcular).

Nota: Si en lugar de insertar numéricamente las diferentes coordenadas del transmisor y el área de cobertura (ya sea rectangular o circular), se desea seleccionar gráficamente con el ratón sobre el grid mostrado en pantalla por *RADIOGIS*, mediante un proceso de cálculo, *RADIOGIS* extraerá del área seleccionada gráficamente, las variables numéricas pasadas como argumentos con las que el programa realizará el cálculo de las pérdidas. Estos argumentos serán: x*llcorner*, *yllcorner* (coordenadas del extremo inferior izquierdo del cuadrado seleccionado sobre el MDT), *nfils* y *ncols* (número de filas y columnas hacia arriba y derecha respectivamente a partir del extremo inferior comentado, sobre las que se desea calcular las pérdidas).

Así pues, el funcionamiento del código es exactamente igual para ambos tipos de áreas. La diferencia que establece este buble es que si se ha seleccionado el área circular, tras levantar el *perfil* y *vector_distancias* en el método *main*, en cada una de las iteraciones por las que se llame al método *CalculaAtenuacion*, dentro de éste, en caso que la distancia sobre el suelo en metros entre el transmisor y el receptor (*R* previamente calculado) sea mayor que el radio de la circunferencia (*radio*), se entiende que dicha celda está fuera de la circunferencia de cálculo y se procede a devolver el valor nulo *-9999*, previamente habiendo liberado el espacio de memoria reservado para

los diferentes vectores creados en la actual llamada. Una vez concluido, se pasará a la siguiente llamada a método, y así hasta que se hayan recorrido los vectores *perfil* y *vector_distancias* en sus totalidades. Tras esto, si aun no ha finalizado el cálculo de pérdidas, se creará un nuevo *perfil* y *vector_distancias* en el método *main*, y todo el proceso se repetirá hasta recorrer el grid en su totalidad.

Con esta explicación finaliza el punto actual del proyecto, puesto que el resto de código que forma parte del fichero "calculaatenuacion.cpp" es el modelo UITR-526 de predicción de pérdidas (explicado teóricamente en el apartado 3.1.2.), y en él no se ha realizado prácticamente ninguna modificación, con la salvedad de sustituir las variables *perfil* y *vector_distancias* por *perfil_actual* y *vector_distancias_actual* respectivamente.

Nota: finalizadas todas las variaciones realizadas en Borland C++, se compila el fichero, y se creará automáticamente el ejecutable "UITRB.exe". Este archivo ejecutable resultante es el que se copia en la carpeta "Ejecutables" de RADIOGIS, y el que realizará los cálculos de pérdidas cuando se seleccione el modelo UITR-526.

3.5. RESULTADOS.

3.5.1. Introducción.

A continuación se explicarán los pasos a realizar en *RADIOGIS* para obtener una determinada cobertura a partir del método de predicción para entornos rurales con el que se está trabajando, es decir, UITR-526. El primer paso es cargar el grid sobre el que realizará el cálculo; en este caso es el de la Región de Murcia, con resolución 200 metros ("mdt_200grid.aux").

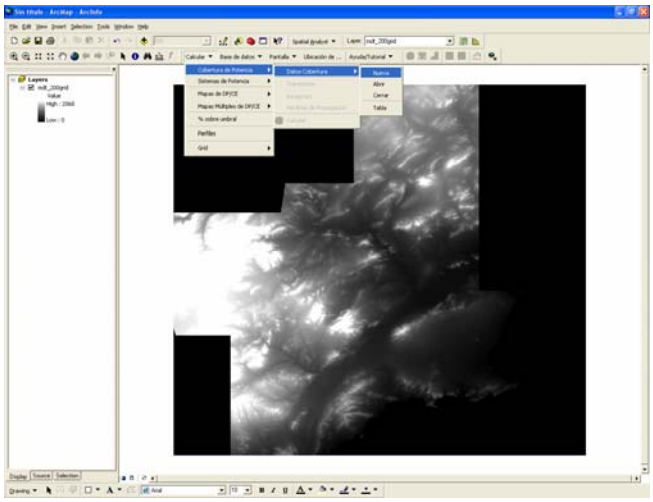


Figura 3.10. Creación de una cobertura rural.

Se selecciona el cálculo de una nueva cobertura radieléctrica, y se le da nombre a ésta. También es posible introducir comentarios sobre la cobertura. En la parte inferior se muestra una lista con las posibles coberturas existentes.

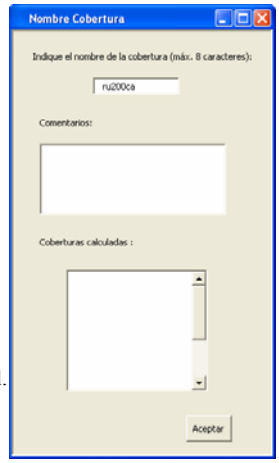


Figura 3.11. Creación de una cobertura rural. Nombre cobertura.

Tras ello, se introducen los parámetros radioeléctricos del transmisor (PIRE, potencia transmitida, ganancia o pérdidas) y receptor (ganancia, pérdidas, sensibilidad).

Figura 3.12. Creación de una cobertura rural. Transmisión y Recepción.

A partir de este momento, se debe completar el formulario "Pérdidas de propagación", que está formado por cuatro pestañas desplegables: "estación base", "estación móvil", "parámetros comunes" y "zona de cálculo".

1. "Estación base".

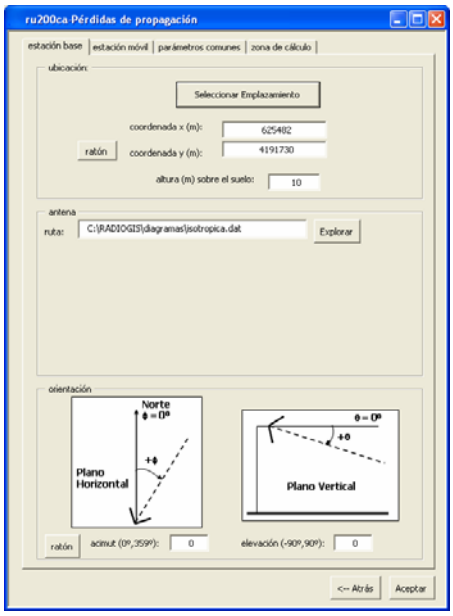


Figura 3.13. Creación de una cobertura rural. Pérdidas de propagación. Estación base.

Se seleccionan las coordenadas en metros (siguiendo el sistema internacional) del emplazamiento del transmisor, y la altura en metros de la antena sobre el suelo.

Existe la posibilidad de cargar un emplazamiento creado anteriormente, o seleccionarlo directamente sobre el mapa que se está visualizando en la pantalla principal de *ArcMap*.

Además, se especifica el tipo de antena (isotrópica, omnidireccional, sectorizada) especificada en un archivo .dat, y el ángulo en acimut y en elevación.

El fichero de extensión ".dat" que define el diagrama de elevación está formado por 360 filas y 181 columnas, cuyos valores contienen la ganancia en dB´s de la antena.

Las filas contienen el diagrama de radiación en acimut, correspondiendo cada fila a un grado en el diagrama. La primera fila es la correspondiente a la dirección norte, y las siguientes contienen el valor del diagrama en cada ángulo entero tomado a partir del norte en sentido de las agujas del reloj. *[5]*

Las columnas contienen el diagrama de radiación en elevación. Cada columna corresponde a un grado en el diagrama, correspondiendo la primera columna a la dirección perpendicular al suelo y la última a la dirección hacia el cielo. *[5]*

Así queda definido el diagrama de radiación de la antena en todas las direcciones con una resolución de un grado. Obsérvese una pequeña porción de este tipo de fichero como ejemplo (el fichero completo consta de 360 filas e ídem columnas):

```
-2.5090000e+001 -2.5090000e+001 -2.4700000e+001 -2.4350000e+001 -2.4030000e+001 ….
-2.3720000e+001 -2.3440000e+001 -2.3200000e+001 -2.3020000e+001 -2.2940000e+001 ….
-2.2930000e+001 -2.2970000e+001 -2.3000000e+001 -2.2960000e+001 -2.2850000e+001 ….
-2.2780000e+001 -2.2890000e+001 -2.3310000e+001 -2.4030000e+001 -2.4880000e+001 ….
-2.5690000e+001 -2.6240000e+001 -2.6380000e+001 -2.6560000e+001 -2.7240000e+001 ….
-2.9030000e+001 -3.2510000e+001 -3.6450000e+001 -3.9360000e+001 -3.9780000e+001 ….
-3.7410000e+001 -3.3600000e+001 -2.9840000e+001 -2.7570000e+001 -2.6610000e+001 ….
-2.6270000e+001 -2.6180000e+001 -2.5920000e+001 -2.5160000e+001 -2.4350000e+001 ….
-2.4130000e+001 -2.5170000e+001 -2.8690000e+001 -3.3730000e+001 -3.8200000e+001 ….
-4.0000000e+001 -3.7690000e+001 -3.2690000e+001 -2.7150000e+001 -2.3200000e+001 ….
                                    ……
```

Figura 3.14. Porción de fichero del diagrama de radiación de una antena.

2. "Estación móvil".

Los mismos pasos se siguen para definir los parámetros de la estación móvil, salvo la diferencia de no reflejar unas coordenadas de recepción fijas, ya que se calcularán las péridas en múltiples celdas (cuanto mayor sea el área de cobertura, y menor la resolución del MDT, más receptores se considerarán).

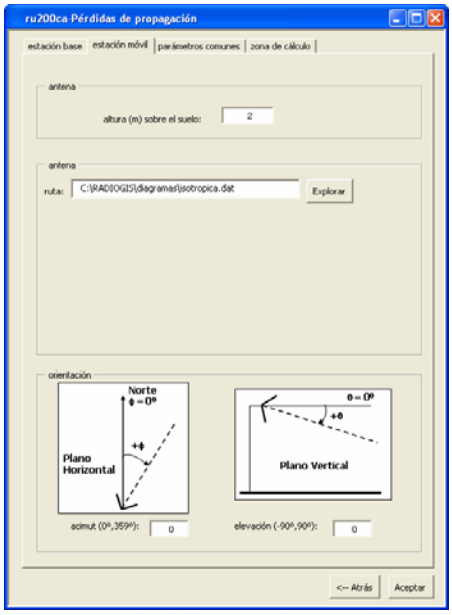
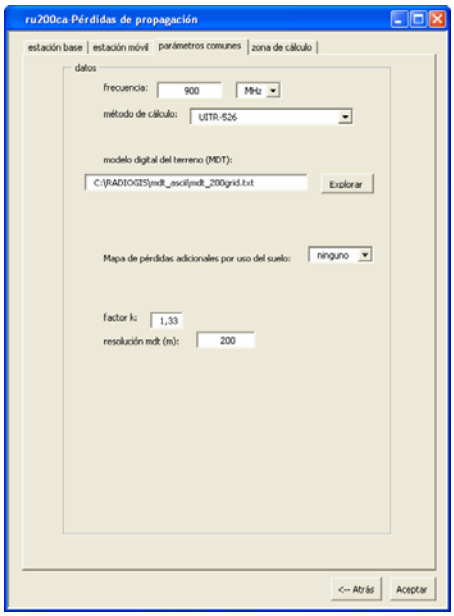


Figura 3.15. Creación de una cobertura rural. Pérdidas de propagación. Estación móvil.

3. "Parámetros comunes".

Esta pestaña es la que más difiere con respecto al cálculo de pérdidas para terreno urbano.

Se elige la frecuencia del radioenlace y el método de cálculo para predicción de pérdidas, en este caso UITR-526, además, se debe especificar el modelo digital del terreno en formato ASCII, es decir, el archivo de texto en el que están almacenadas las alturas, coordenadas y resolución ("mdt_200grid.txt" en este caso), que se corresponden con el grid del que se dispone en pantalla.



También se puede cargar un mapa en el que se reflejen discontinuidades adicionales sobre el terreno considerado.

Figura 3.16. Creación de una cobertura rural.
Pérdidas de propagación. Parámetros comunes.

Por último, se fija el factor "k", y la resolución del MDT, la cual debe coincidir con el respectivo grid ("mdt_200grid.aux").

4. "Zona de cálculo".

Finalmente, se puede calcular las pérdidas en un a´rea rectangular o circular.

Para el cálculo sobre un área circular, se fijan las coordenadas del centro del círculo (las cuales coincidirán con las del emplazamiento del transmisor), y el radio en metros de dicha área.

Cuando el área seleccionada es rectangular, se fijan los extremos de este rectángulo.

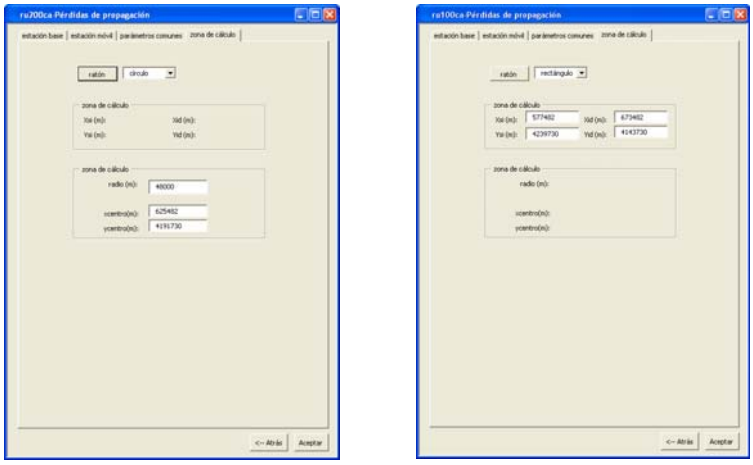


Figura 3.17. Creación de una cobertura rural. Pérdidas de propagación. Zonas de cálculo rectangular y circular.

Además, ambas áreas pueden ser seleccionadas automáticamenete con el ratón sobre el mapa reflejado sobre *ArcMap*. Ya sea con la selección automática o no, el transmisor siempre estará en el cento del área circular; sin embargo, en el área rectangular no tiene por qué ser necesriamente así.

Introducidos todos los datos, el programa ya se encuentra en disposición de realizar el cálculo de la cobertura. Al pulsar aceptar, se activará el ejecutable "UITRB.exe".

Nota: Todas las simulaciones se han realizado en un mismo ordenador con procesador Intel Pentium 4 de 2,40 GHz y 512 MB de RAM. El sistema operativo es Microsoft Windows XP Professional.

3.5.2. MDT resolución 200.

3.5.2.1. Área rectangular.

MDT200GRID											
LADO (Km)	ÁREA (Km^2)	TIEMPOS									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
4,80	23,04	0,015	0,016	0,001	6,250	0	0,0	0	0,0	0	0,0
9,60	92,16	0,078	0,094	0,016	17,021	0	0,1	0	0,1	0	0,0
14,40	207,36	0,156	0,265	0,109	41,132	0	0,2	0	0,3	0	0,1
19,20	368,64	0,328	0,562	0,234	41,637	0	0,3	0	0,6	0	0,2
24,00	576,00	0,516	1,000	0,484	48,400	0	0,5	0	1,0	0	0,5
28,80	829,44	0,750	1,562	0,812	51,985	0	0,8	0	1,6	0	0,8
33,60	1128,96	1,109	2,359	1,250	52,989	0	1,1	0	2,4	0	1,3
38,40	1474,56	1,531	3,437	1,906	55,455	0	1,5	0	3,4	0	1,9
43,20	1866,24	2,125	4,813	2,688	55,849	0	2,1	0	4,8	0	2,7
48,00	2304,00	2,719	6,484	3,765	58,066	0	2,7	0	6,5	0	3,8
52,80	2787,84	3,625	8,641	5,016	58,049	0	3,6	0	8,6	0	5,0
57,60	3317,76	4,671	11,312	6,641	58,708	0	4,7	0	11,3	0	6,6
62,40	3893,76	5,953	14,203	8,250	58,086	0	6,0	0	14,2	0	8,3
67,20	4515,84	7,468	17,828	10,360	58,111	0	7,5	0	17,8	0	10,4
72,00	5184,00	9,422	22,031	12,609	57,233	0	9,4	0	22,0	0	12,6
76,80	5898,24	11,454	26,922	15,468	57,455	0	11,5	0	26,9	0	15,5
81,60	6658,56	13,953	32,563	18,610	57,151	0	14,0	0	32,6	0	18,6
86,40	7464,96	17,000	39,172	22,172	56,602	0	17,0	0	39,2	0	22,2
91,20	8317,44	20,484	46,563	26,079	56,008	0	20,5	0	46,6	0	26,1
96,00	9216,00	24,313	54,625	30,312	55,491	0	24,3	0	54,6	0	30,3

Tabla 3.6. Tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 200 metros.

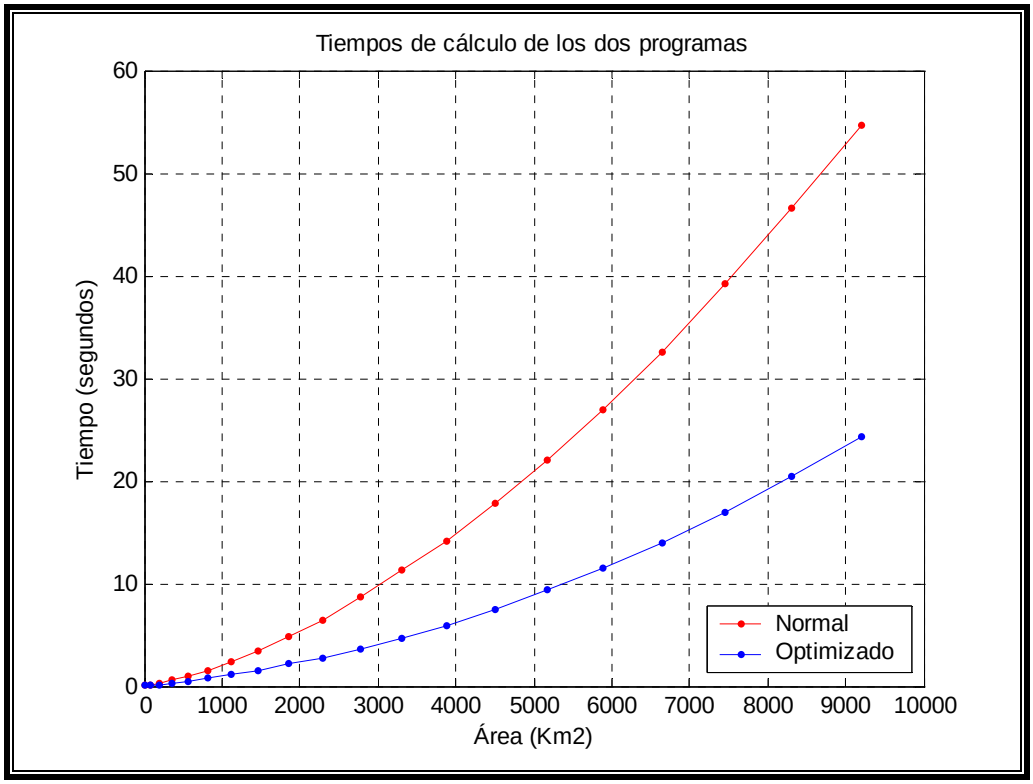


Figura 3.18. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 200 metros.

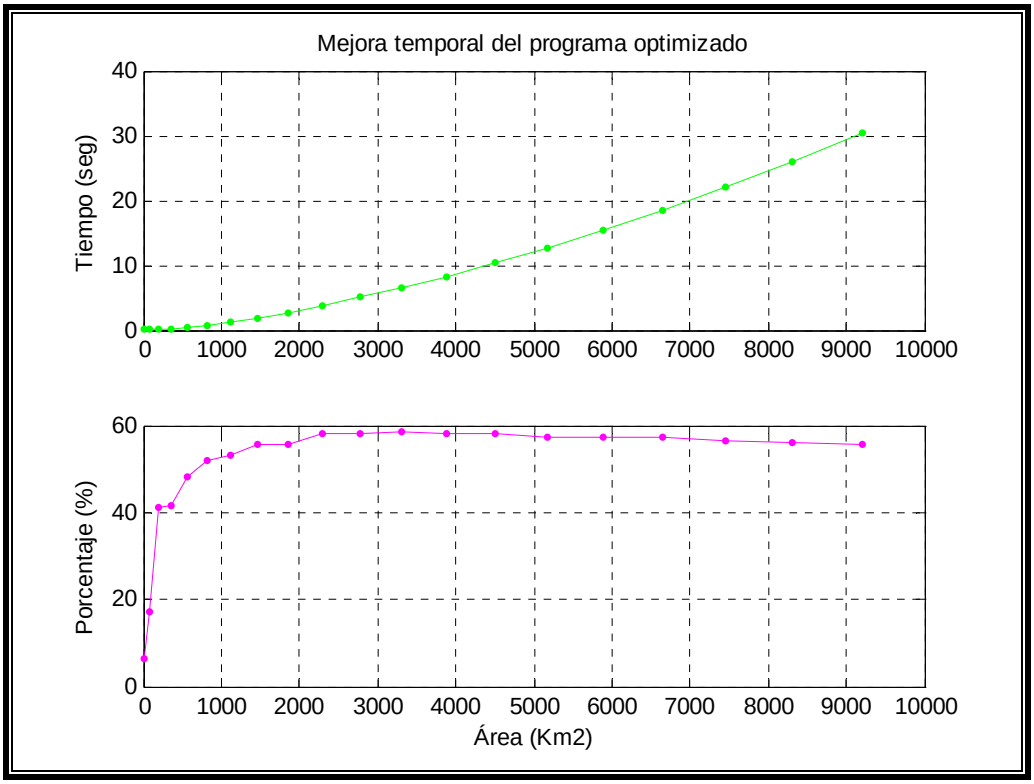


Figura 3.19. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 200 metros.

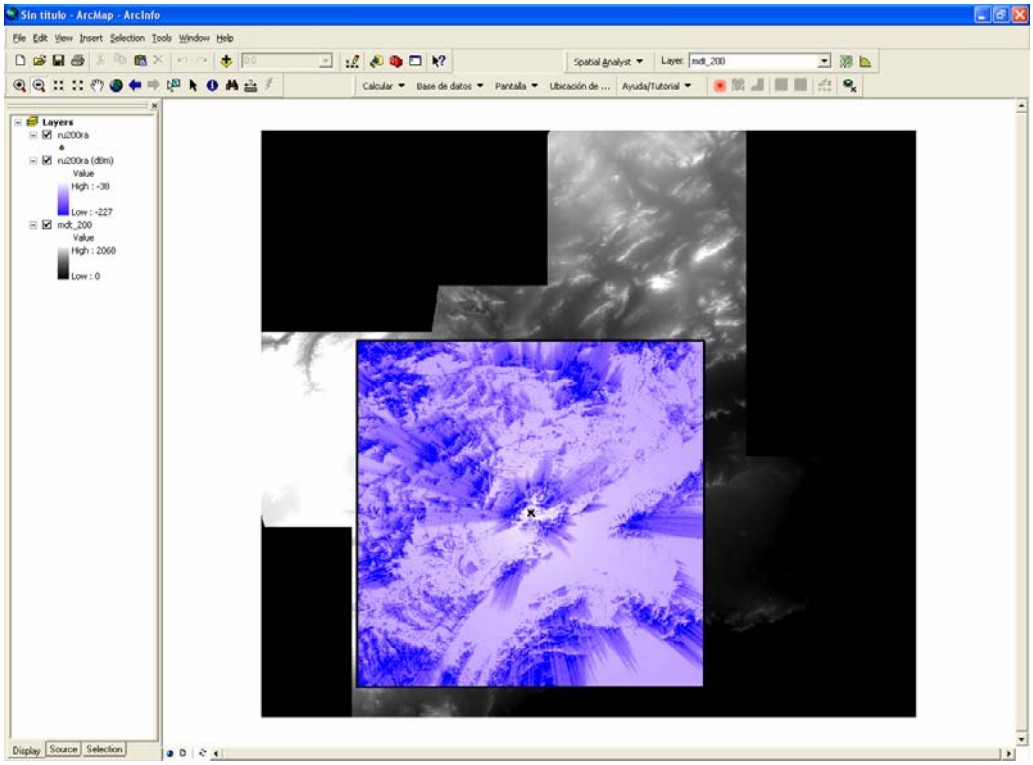


Figura 3.20. Pérdidas calculadas por el programa modificado sobre un área rectangular, para un MDT rural de resolución 200 metros.

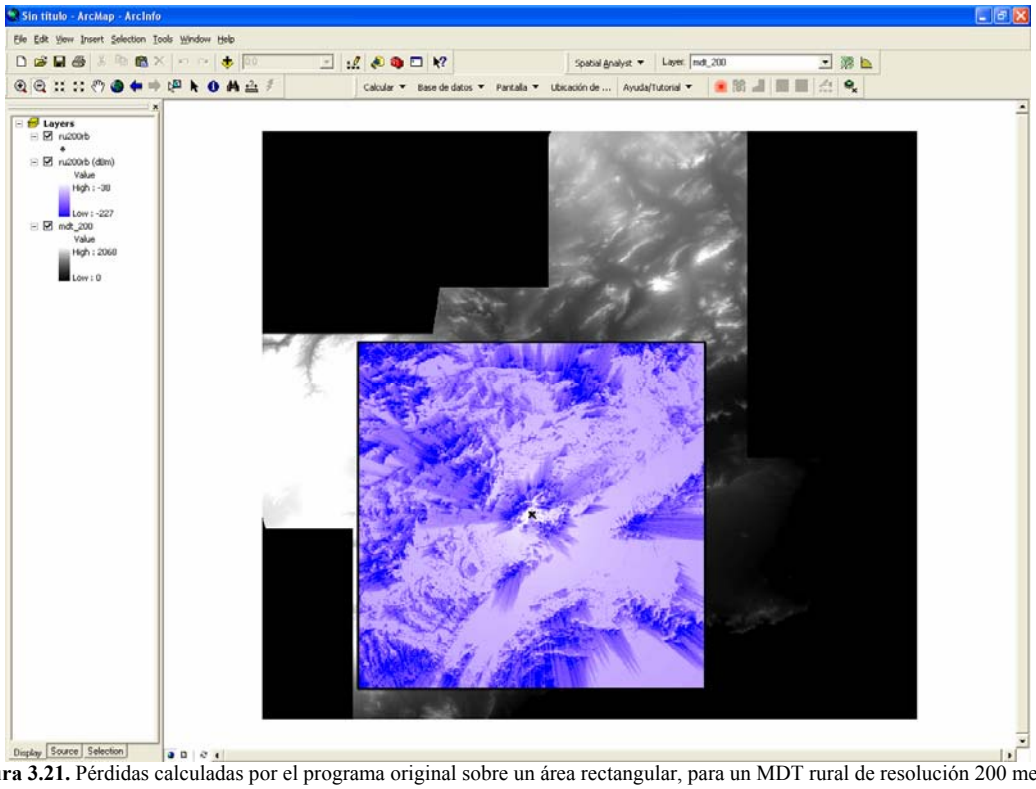


Figura 3.21. Pérdidas calculadas por el programa original sobre un área rectangular, para un MDT rural de resolución 200 metros.

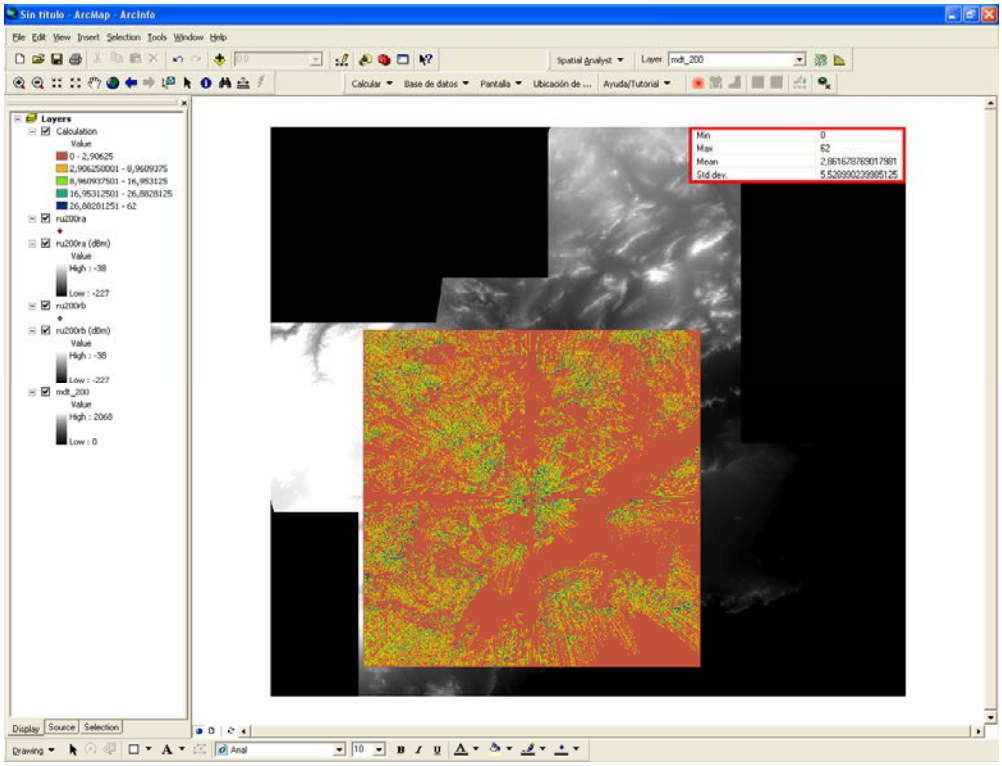


Figura 3.22. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT rural de resolución 200 metros.

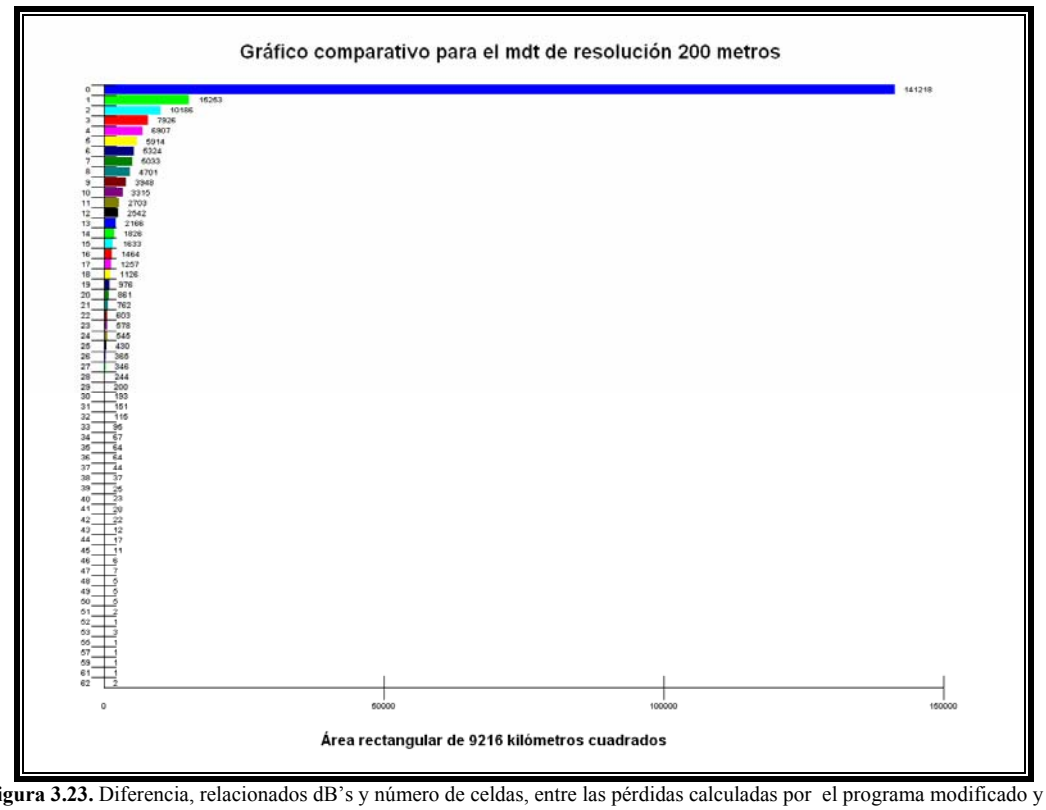


Figura 3.23. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT rural de resolución 200 metros.

3.5.2.2. Área circular.

MDT200GRID											
RADIO (Km)	ÁREA (Km^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
2,40	18,10	0,015	0,016	0,001	6,250	0	0,0	0	0,0	0	0,0
4,80	72,38	0,047	0,062	0,015	24,194	0	0,0	0	0,1	0	0,0
7,20	162,86	0,125	0,187	0,062	33,155	0	0,1	0	0,2	0	0,1
9,60	289,53	0,265	0,406	0,141	34,729	0	0,3	0	0,4	0	0,1
12,00	452,39	0,421	0,719	0,298	41,446	0	0,4	0	0,7	0	0,3
14,40	651,44	0,625	1,125	0,500	44,444	0	0,6	0	1,1	0	0,5
16,80	886,68	0,890	1,703	0,813	47,739	0	0,9	0	1,7	0	0,8
19,20	1158,12	1,219	2,438	1,219	50,000	0	1,2	0	2,4	0	1,2
21,60	1465,74	1,625	3,391	1,766	52,079	0	1,6	0	3,4	0	1,8
24,00	1809,56	2,125	4,547	2,422	53,266	0	2,1	0	4,5	0	2,4
26,40	2189,56	2,750	6,094	3,344	54,874	0	2,8	0	6,1	0	3,3
28,80	2605,76	3,484	7,813	4,329	55,408	0	3,5	0	7,8	0	4,3
31,20	3058,15	4,532	9,890	5,358	54,176	0	4,5	0	9,9	0	5,4
33,60	3546,73	5,609	12,313	6,704	54,447	0	5,6	0	12,3	0	6,7
36,00	4071,50	7,016	15,219	8,203	53,900	0	7,0	0	15,2	0	8,2
38,40	4632,47	8,485	18,531	10,046	54,212	0	8,5	0	18,5	0	10,0
40,80	5229,62	10,266	22,328	12,062	54,022	0	10,3	0	22,3	0	12,1
43,20	5862,97	12,344	26,735	14,391	53,828	0	12,3	0	26,7	0	14,4
45,60	6532,50	14,719	31,687	16,968	53,549	0	14,7	0	31,7	0	17,0
48,00	7238,23	17,313	37,094	19,781	53,327	0	17,3	0	37,1	0	19,8

Tabla 3.7. Tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 200 metros.

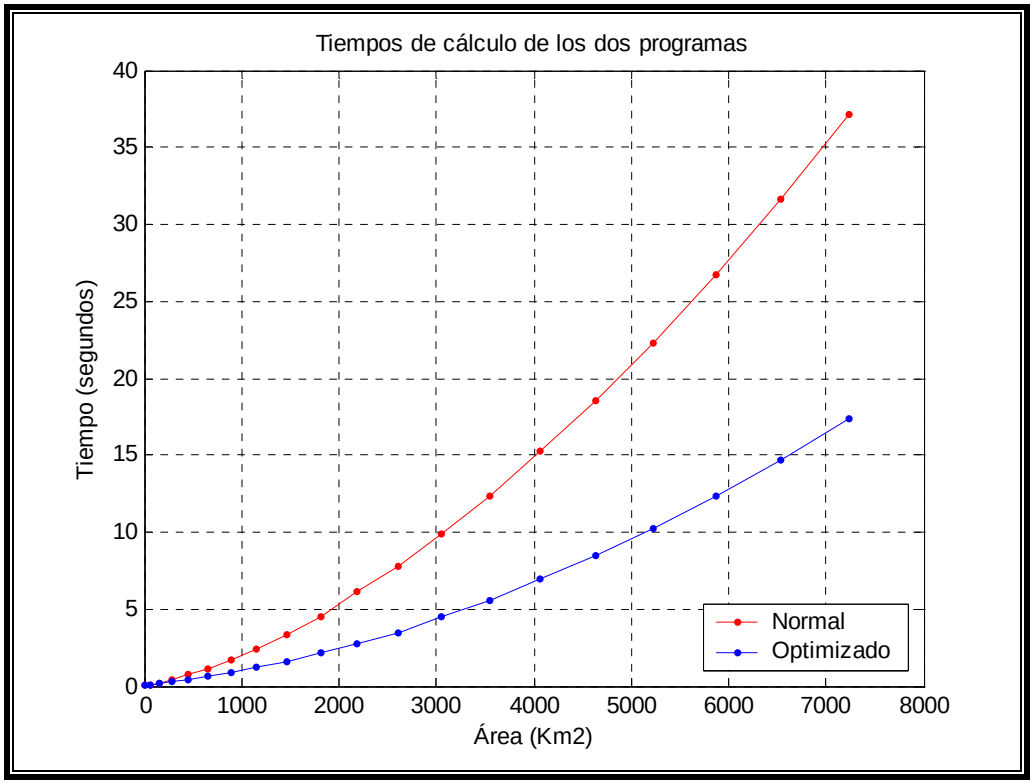


Figura 3.24. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 200 metros.

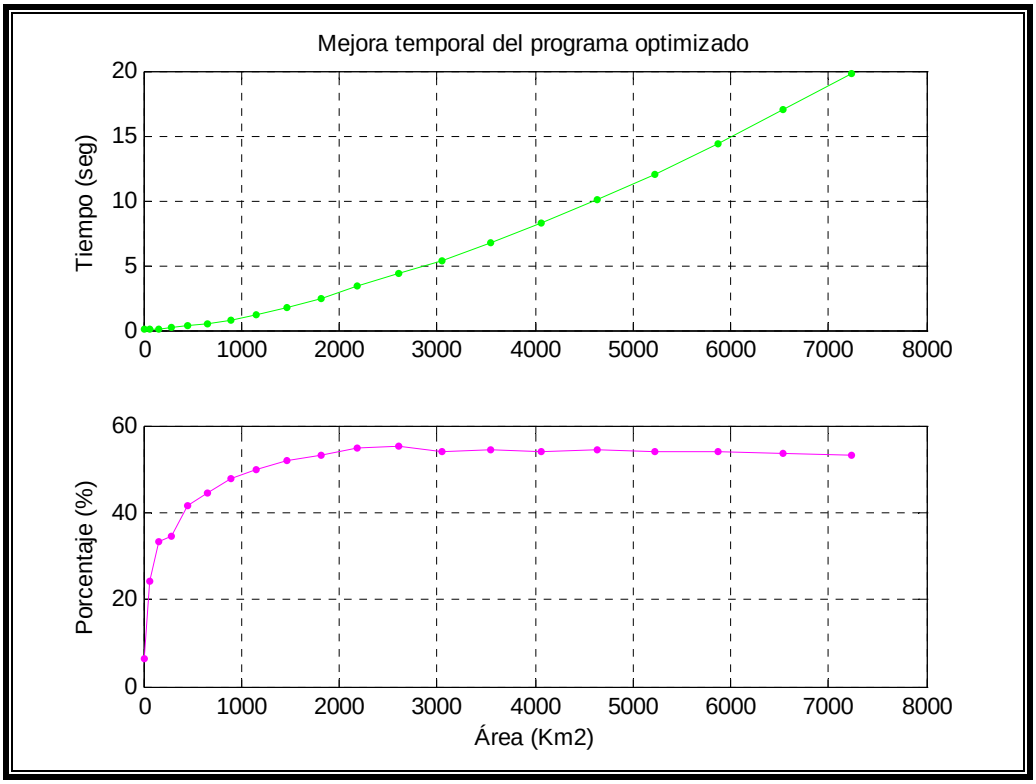


Figura 3.25. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 200 metros.

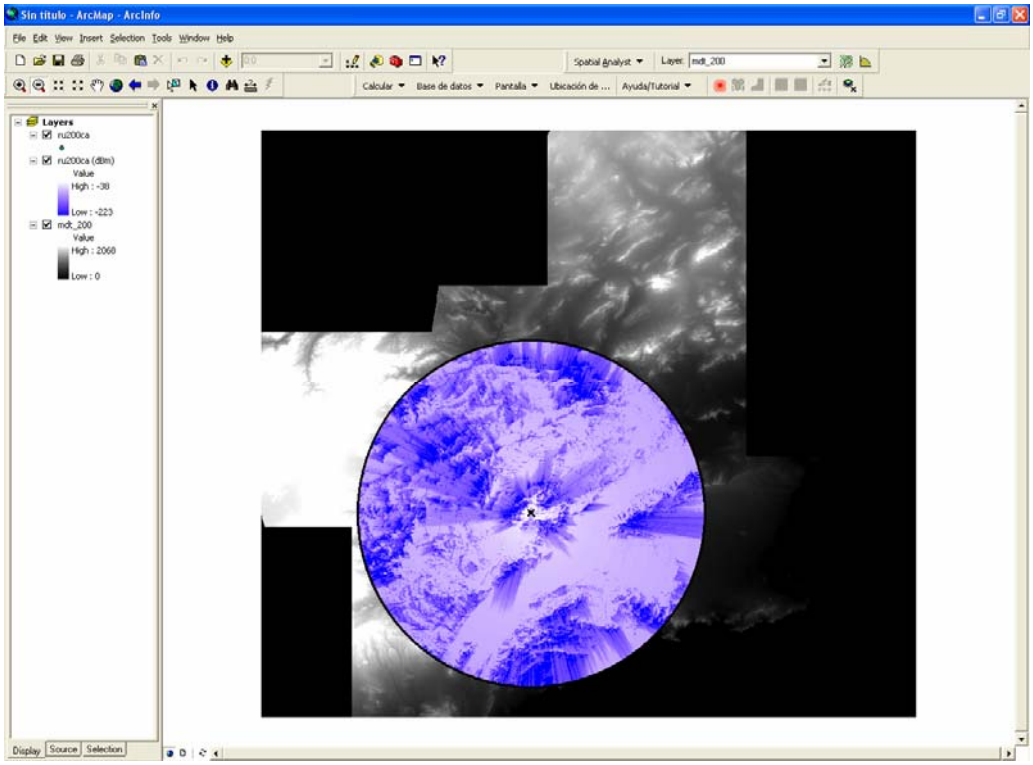


Figura 3.26. Pérdidas calculadas por el programa modificado sobre un área circular, para un MDT rural de resolución 200 metros.

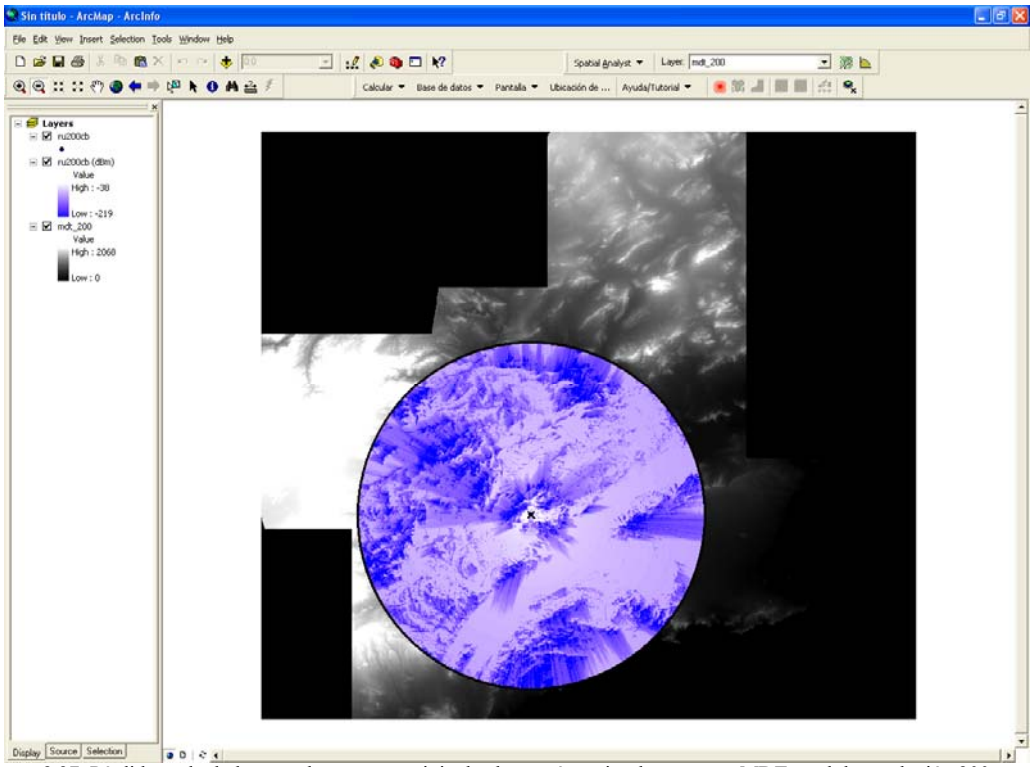


Figura 3.27. Pérdidas calculadas por el programa original sobre un área circular, para un MDT rural de resolución 200 metros.

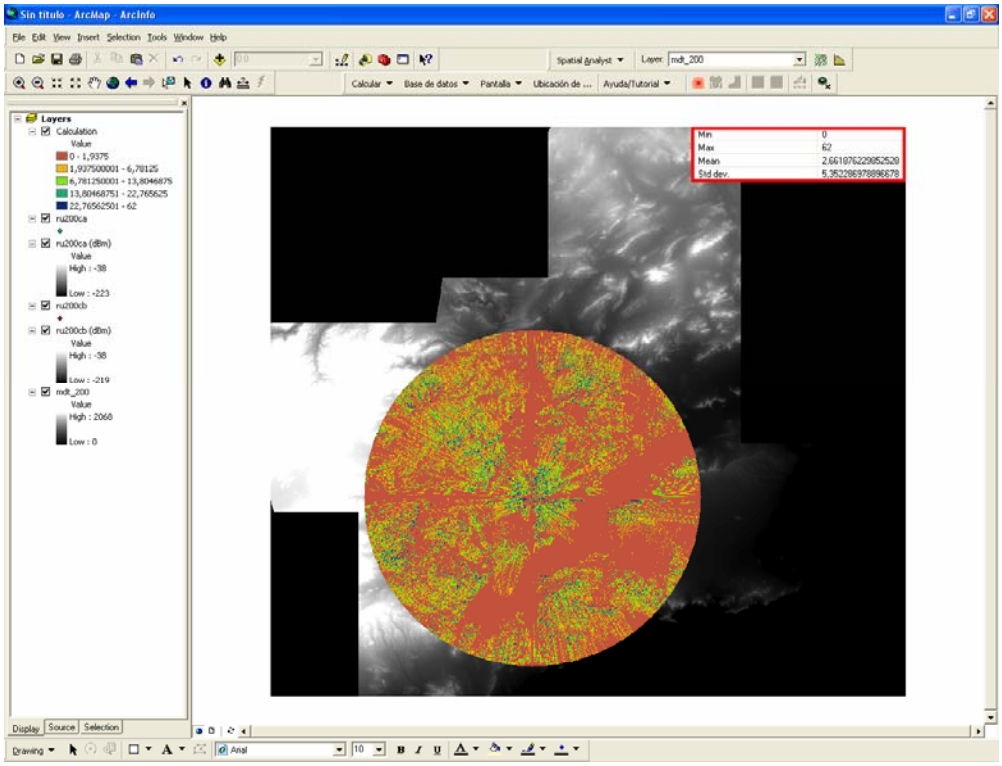


Figura 3.28. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT rural de resolución 200 metros.

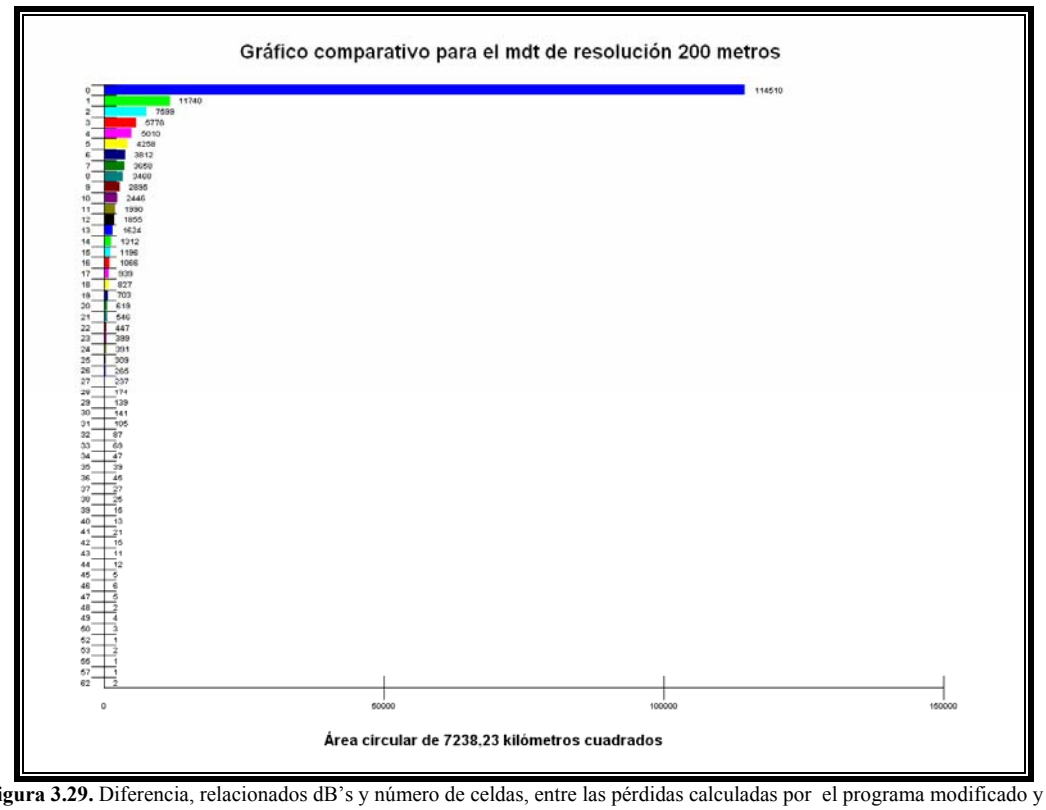


Figura 3.29. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT rural de resolución 200 metros.

3.5.3. MDT resolución 100.

3.5.3.1. Área rectangular.

MDT100GRID											
LADO (Km)	ÁREA (Km^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
4,80	23,04	0,078	0,079	0,001	1,266	0	0,1	0	0,1	0	0,0
9,60	92,16	0,422	0,641	0,219	34,165	0	0,4	0	0,6	0	0,2
14,40	207,36	1,250	2,046	0,796	38,905	0	1,3	0	2,0	0	0,8
19,20	368,64	2,719	4,656	1,937	41,602	0	2,7	0	4,7	0	1,9
24,00	576,00	4,750	8,515	3,765	44,216	0	4,8	0	8,5	0	3,8
28,80	829,44	7,406	13,968	6,562	46,979	0	7,4	0	14,0	0	6,6
33,60	1128,96	10,969	21,469	10,500	48,908	0	11,0	0	21,5	0	10,5
38,40	1474,56	15,562	31,235	15,673	50,178	0	15,6	0	31,2	0	15,7
43,20	1866,24	21,438	43,718	22,280	50,963	0	21,4	0	43,7	0	22,3
48,00	2304,00	28,781	59,390	30,609	51,539	0	28,8	0	59,4	0	30,6
52,80	2787,84	38,203	79,109	40,906	51,708	0	38,2	1	19,1	0	40,9
57,60	3317,76	49,922	103,078	53,156	51,569	0	49,9	1	43,1	0	53,2
62,40	3893,76	64,359	131,906	67,547	51,208	1	4,4	2	11,9	1	7,5
67,20	4515,84	82,235	165,047	82,812	50,175	1	22,2	2	45,0	1	22,8
72,00	5184,00	101,157	204,969	103,812	50,648	1	41,2	3	25,0	1	43,8
76,80	5898,24	123,281	250,140	126,859	50,715	2	3,3	4	10,1	2	6,9
81,60	6658,56	150,828	301,797	150,969	50,023	2	30,8	5	1,8	2	31,0
86,40	7464,96	180,657	360,828	180,171	49,933	3	0,7	6	0,8	3	0,2
91,20	8317,44	217,516	431,063	213,547	49,540	3	37,5	7	11,1	3	33,5
96,00	9216,00	257,907	506,375	248,468	49,068	4	17,9	8	26,4	4	8,5

Tabla 3.8. Tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 100 metros.

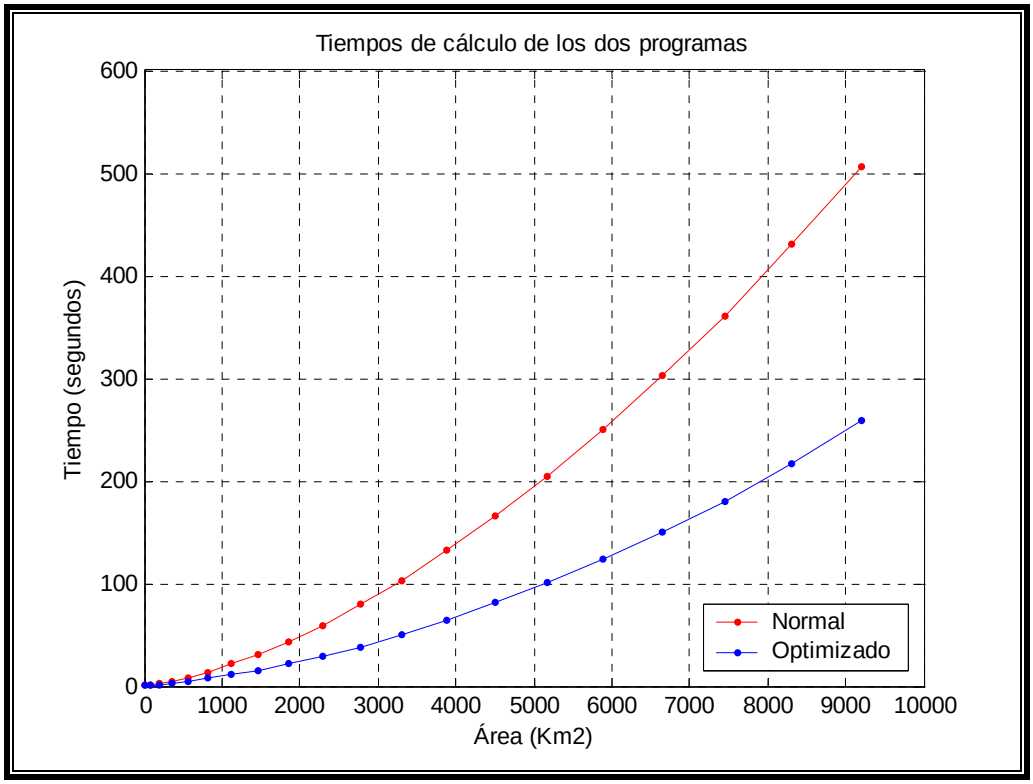


Figura 3.30. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 100 metros.

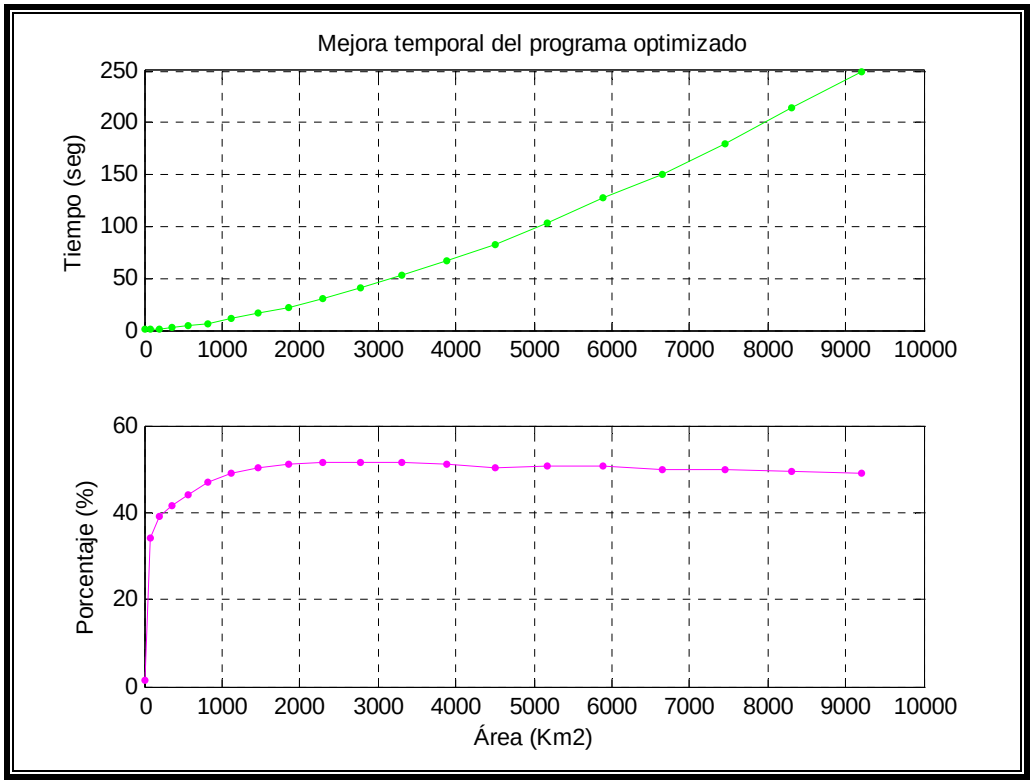


Figura 3.31. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 100 metros.

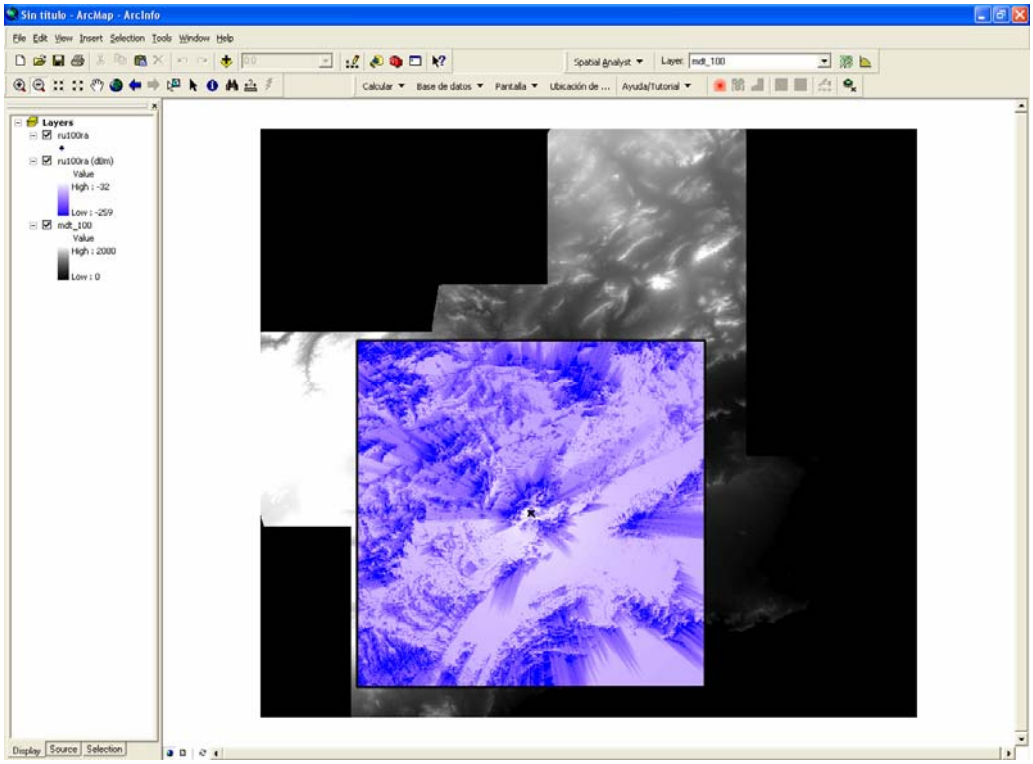


Figura 3.32. Pérdidas calculadas por el programa modificado sobre un área rectangular, para un MDT rural de resolución 100 metros.

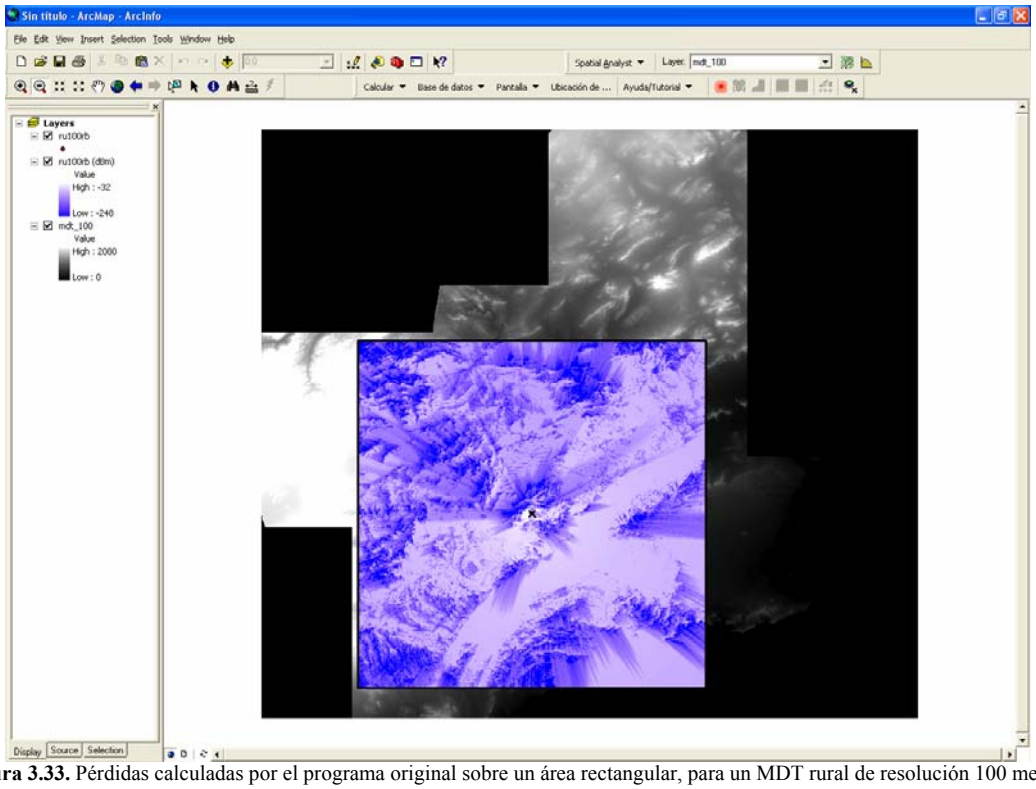


Figura 3.33. Pérdidas calculadas por el programa original sobre un área rectangular, para un MDT rural de resolución 100 metros.

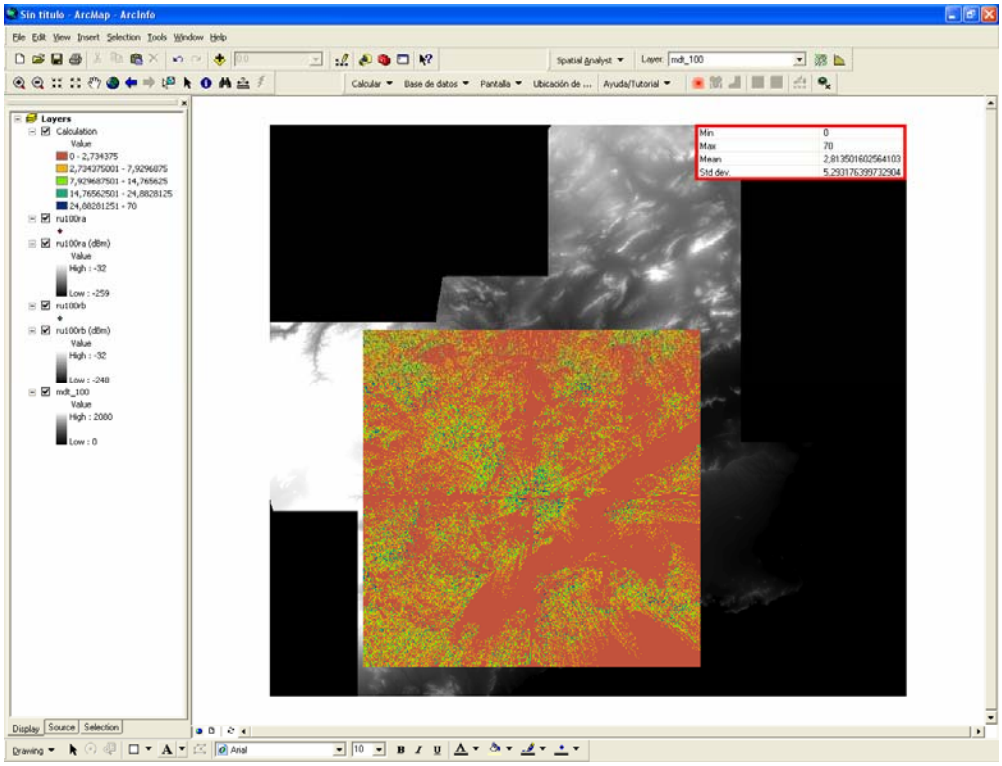


Figura 3.34. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT rural de resolución 100 metros.

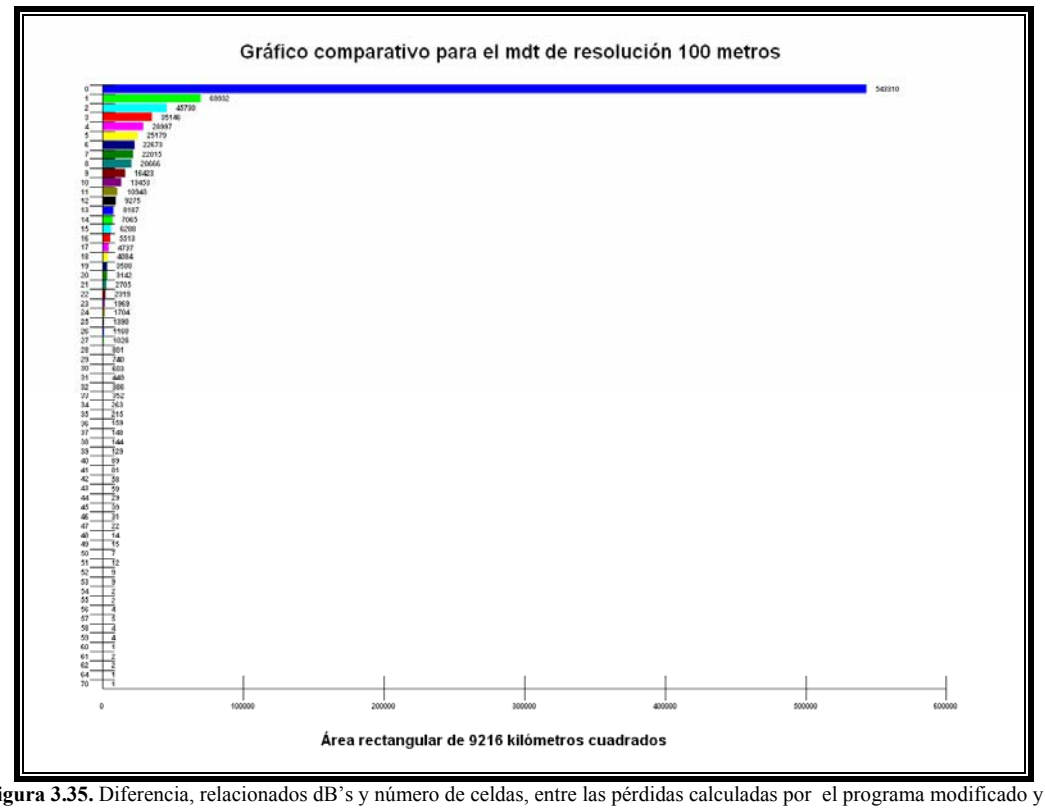


Figura 3.35. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT rural de resolución 100 metros.

3.5.3.2. Área circular.

MDT100GRID											
RADIO (Km)	ÁREA (Km^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
2,40	18,10	0,031	0,062	0,031	50,000	0	0,0	0	0,1	0	0,0
4,80	72,38	0,313	0,453	0,140	30,905	0	0,3	0	0,5	0	0,1
7,20	162,86	0,938	1,422	0,484	34,037	0	0,9	0	1,4	0	0,5
9,60	289,53	2,078	3,281	1,203	36,666	0	2,1	0	3,3	0	1,2
12,00	452,39	3,687	6,203	2,516	40,561	0	3,7	0	6,2	0	2,5
14,40	651,44	5,890	10,015	4,125	41,188	0	5,9	0	10,0	0	4,1
16,80	886,68	8,578	15,203	6,625	43,577	0	8,6	0	15,2	0	6,6
19,20	1158,12	11,953	21,922	9,969	45,475	0	12,0	0	21,9	0	10,0
21,60	1465,74	16,188	30,391	14,203	46,734	0	16,2	0	30,4	0	14,2
24,00	1809,56	21,297	41,109	19,812	48,194	0	21,3	0	41,1	0	19,8
26,40	2189,56	27,609	54,172	26,563	49,035	0	27,6	0	54,2	0	26,6
28,80	2605,76	35,750	70,203	34,453	49,076	0	35,8	1	10,2	0	34,5
31,20	3058,15	45,625	90,703	45,078	49,698	0	45,6	1	30,7	0	45,1
33,60	3546,73	57,719	112,437	54,718	48,665	0	57,7	1	52,4	0	54,7
36,00	4071,50	72,234	140,515	68,281	48,593	1	12,2	2	20,5	1	8,3
38,40	4632,47	88,625	171,953	83,328	48,460	1	28,6	2	52,0	1	23,3
40,80	5229,62	108,765	207,422	98,657	47,563	1	48,8	3	27,4	1	38,7
43,20	5862,97	131,203	247,719	116,516	47,036	2	11,2	4	7,7	1	56,5
45,60	6532,50	154,484	295,953	141,469	47,801	2	34,5	4	56,0	2	21,5
48,00	7238,23	181,484	345,094	163,610	47,410	3	1,5	5	45,1	2	43,6

Tabla 3.9. Tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 100 metros.

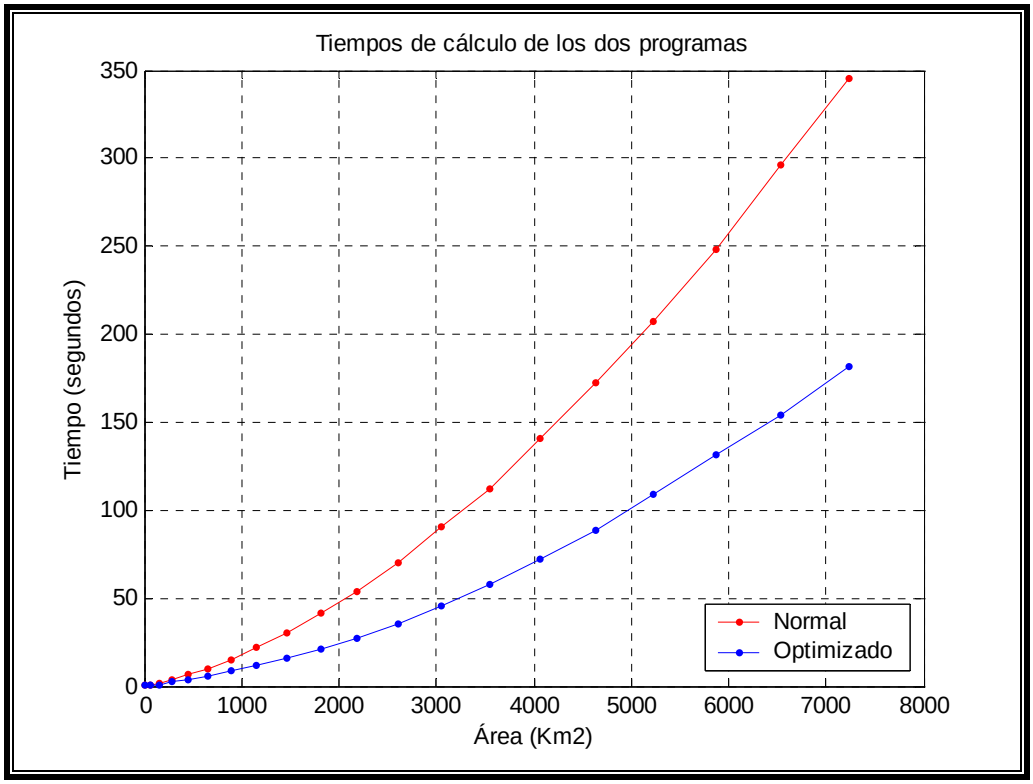


Figura 3.36. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 100 metros.

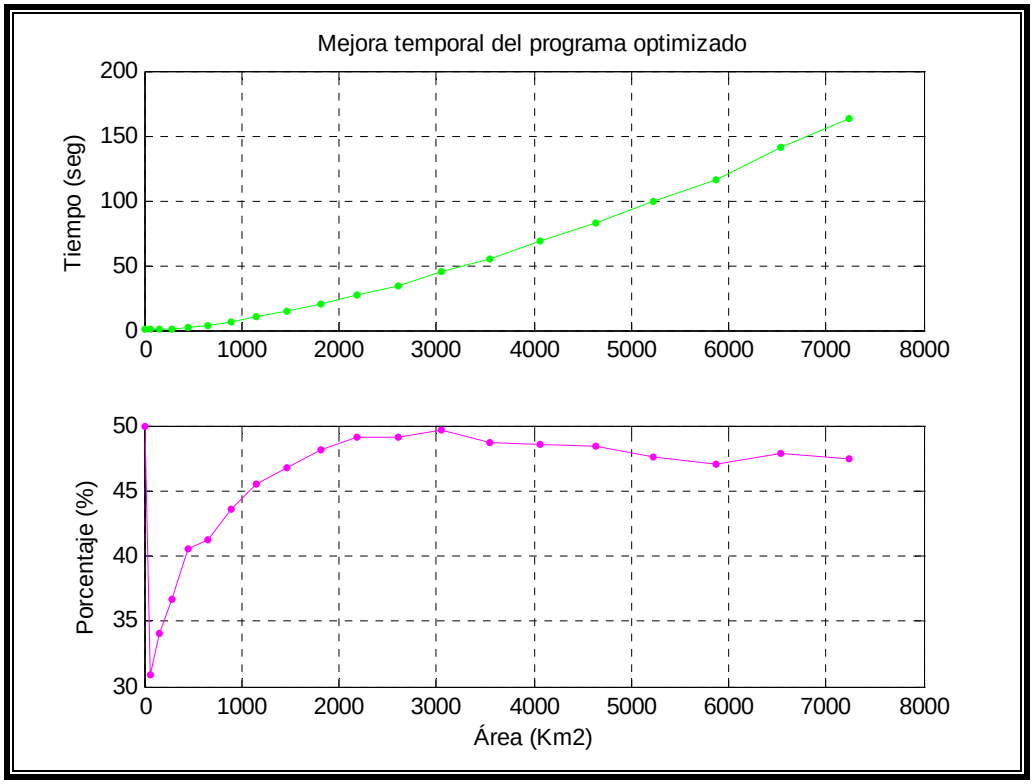


Figura 3.37. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 100 metros.

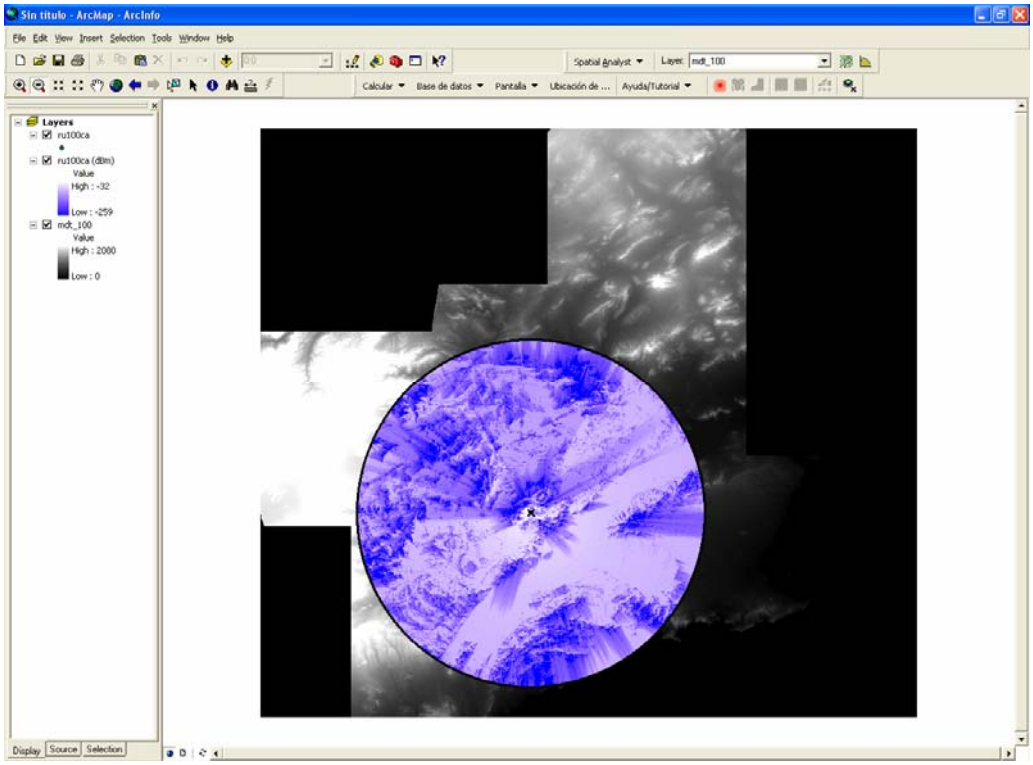


Figura 3.38. Pérdidas calculadas por el programa modificado sobre un área circular, para un MDT rural de resolución 100 metros.

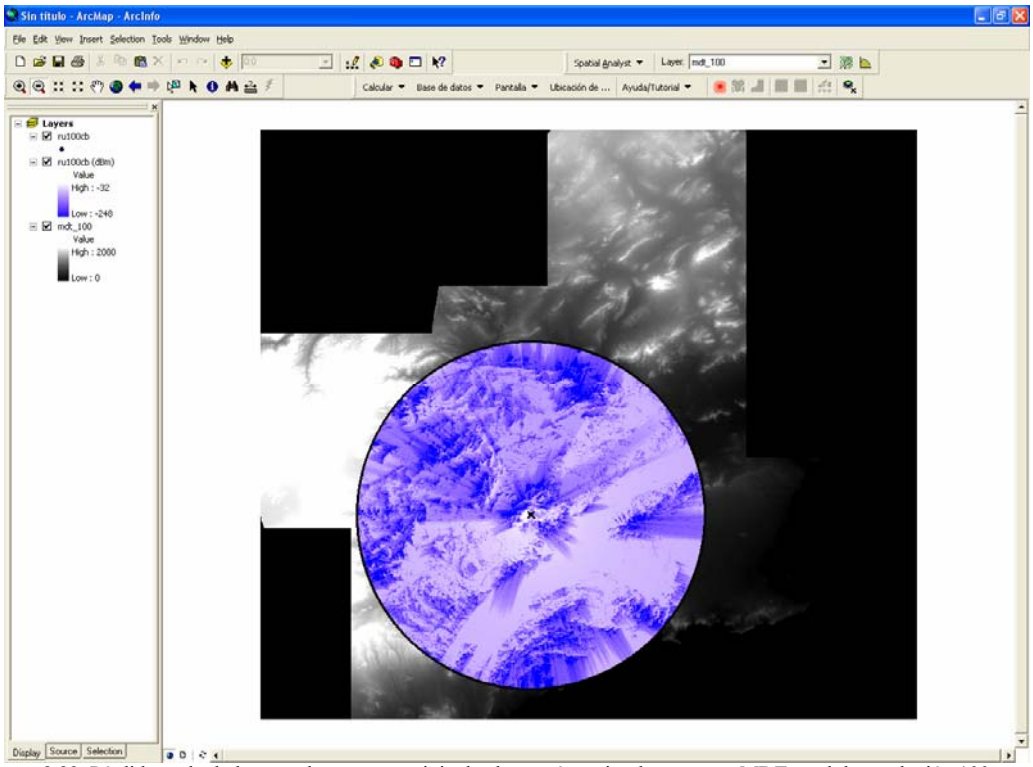


Figura 3.39. Pérdidas calculadas por el programa original sobre un área circular, para un MDT rural de resolución 100 metros.

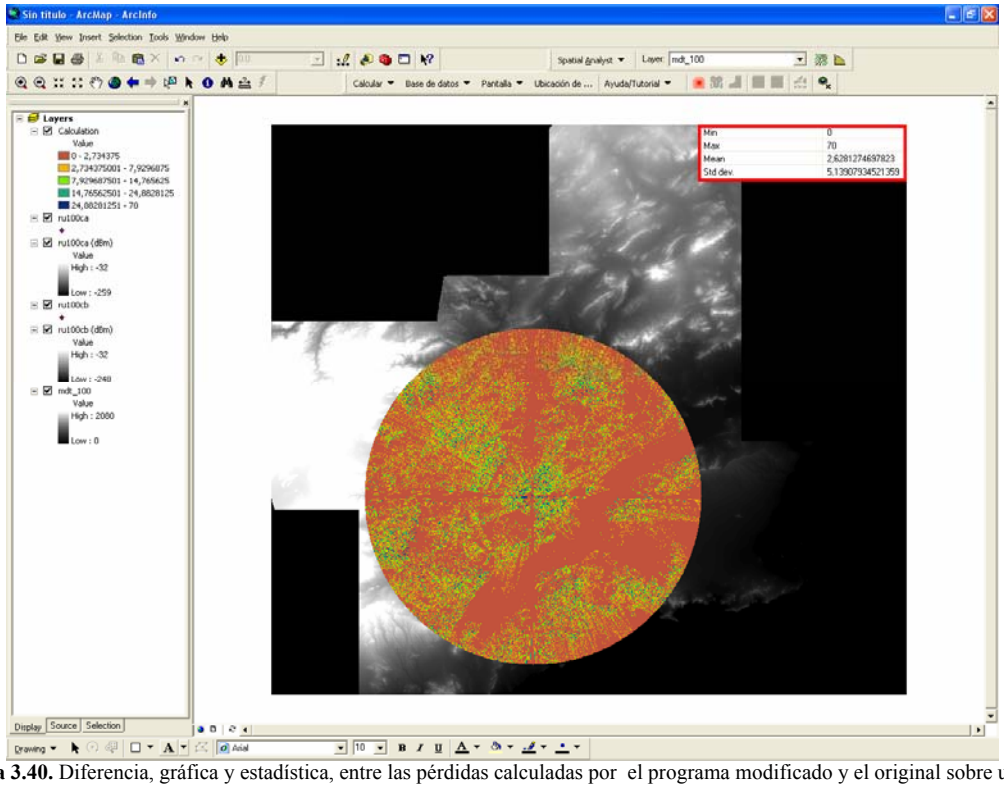


Figura 3.40. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT rural de resolución 100 metros.

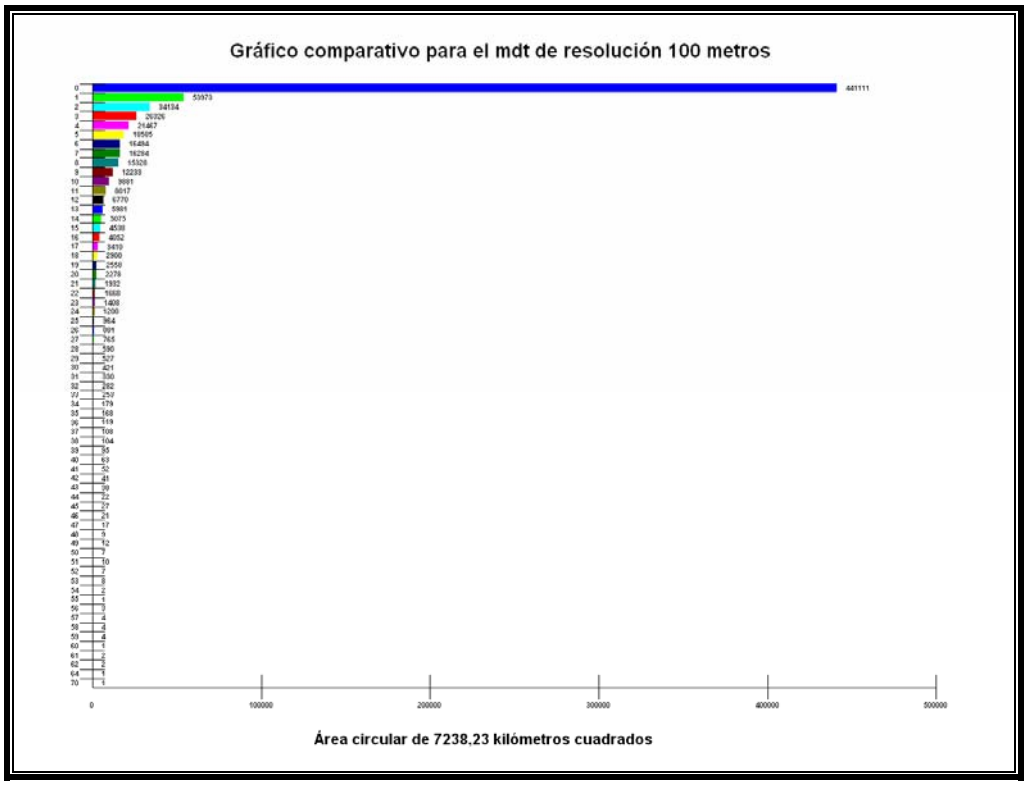


Figura 3.41. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT rural de resolución 100 metros.

3.5.4. MDT resolución 30.

3.5.4.1. Área rectangular.

MDT30GRID											
LADO (Km)	ÁREA (Km^2)	TIEMPO				A		B		B-A	
		A (seg)	B (seg)	B-A (seg)	B-A (%)	min	seg	min	seg	min	seg
4,80	23,04	1,625	2,735	1,110	40,585	0	1,6	0	2,7	0	1,1
9,60	92,16	15,703	24,719	9,016	36,474	0	15,7	0	24,7	0	9,0
14,40	207,36	52,500	83,031	30,531	36,771	0	52,5	1	23,0	0	30,5
19,20	368,64	112,969	186,359	73,390	39,381	1	53,0	3	6,4	1	13,4
24,00	576,00	190,469	336,578	146,109	43,410	3	10,5	5	36,6	2	26,1
28,80	829,44	303,234	552,609	249,375	45,127	5	3,2	9	12,6	4	9,4
33,60	1128,96	462,375	860,625	398,250	46,275	7	42,4	14	20,6	6	38,3
38,40	1474,56	664,640	1256,234	591,594	47,093	11	4,6	20	56,2	9	51,6
43,20	1866,24	921,094	1779,329	858,235	48,234	15	21,1	29	39,3	14	18,2
48,00	2304,00	1283,843	2426,610	1142,767	47,093	21	23,8	40	26,6	19	2,8

Tabla 3.10. Tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 30 metros.

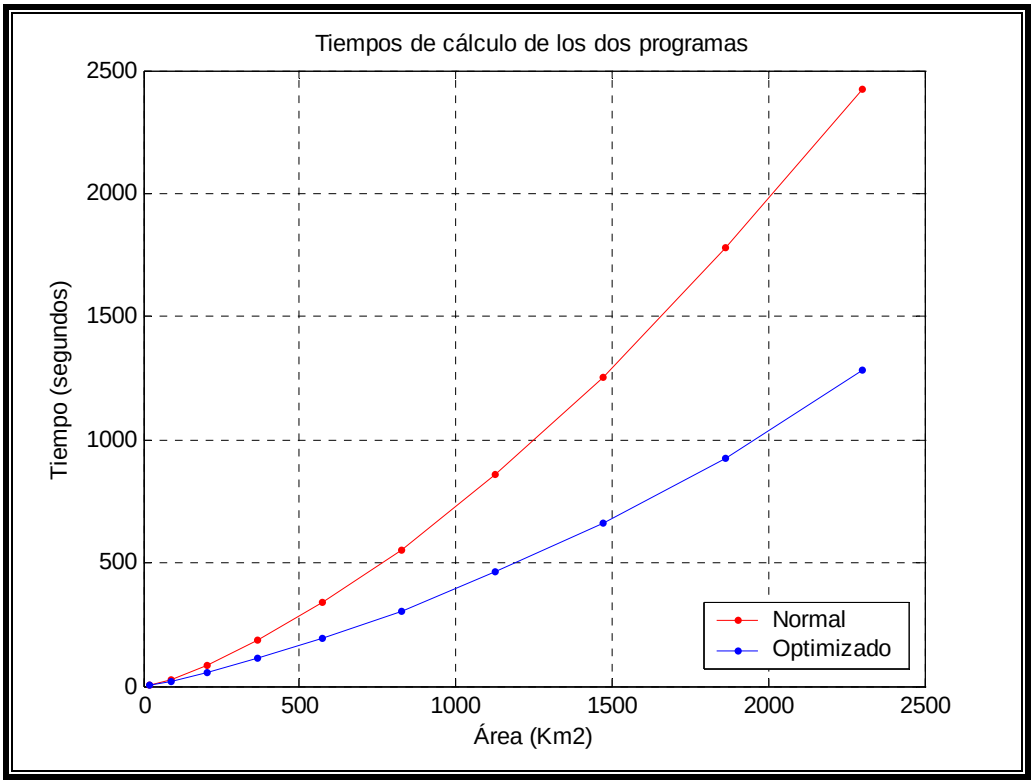


Figura 3.42. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 30 metros.

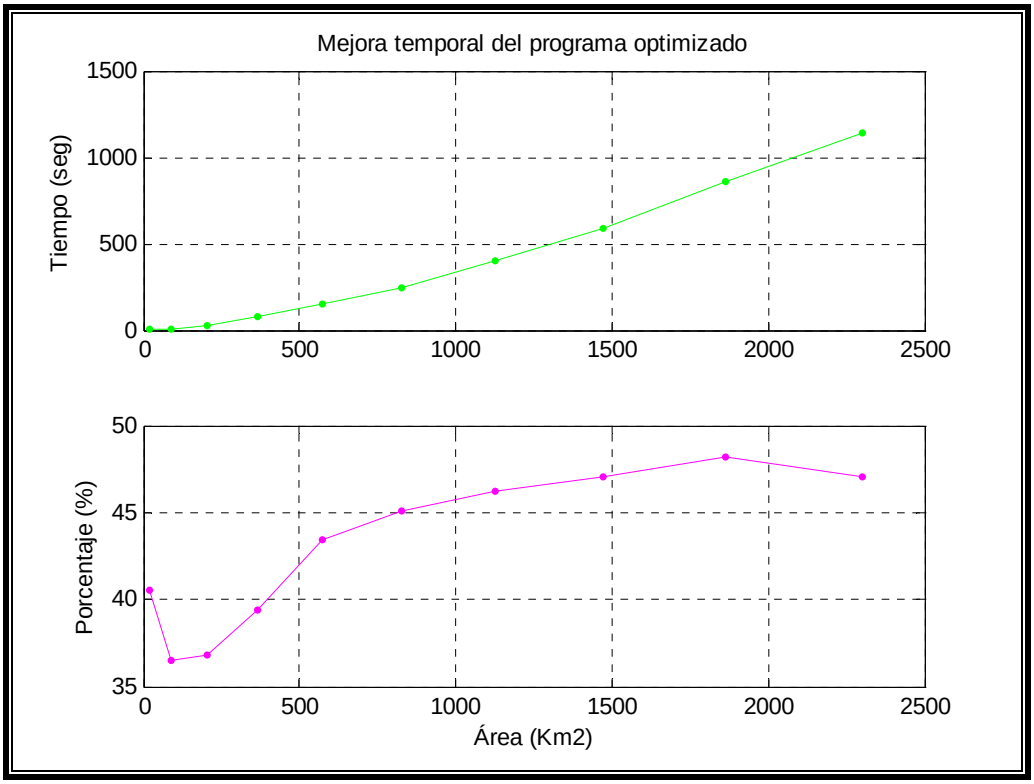


Figura 3.43. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT rural de resolución 30 metros.

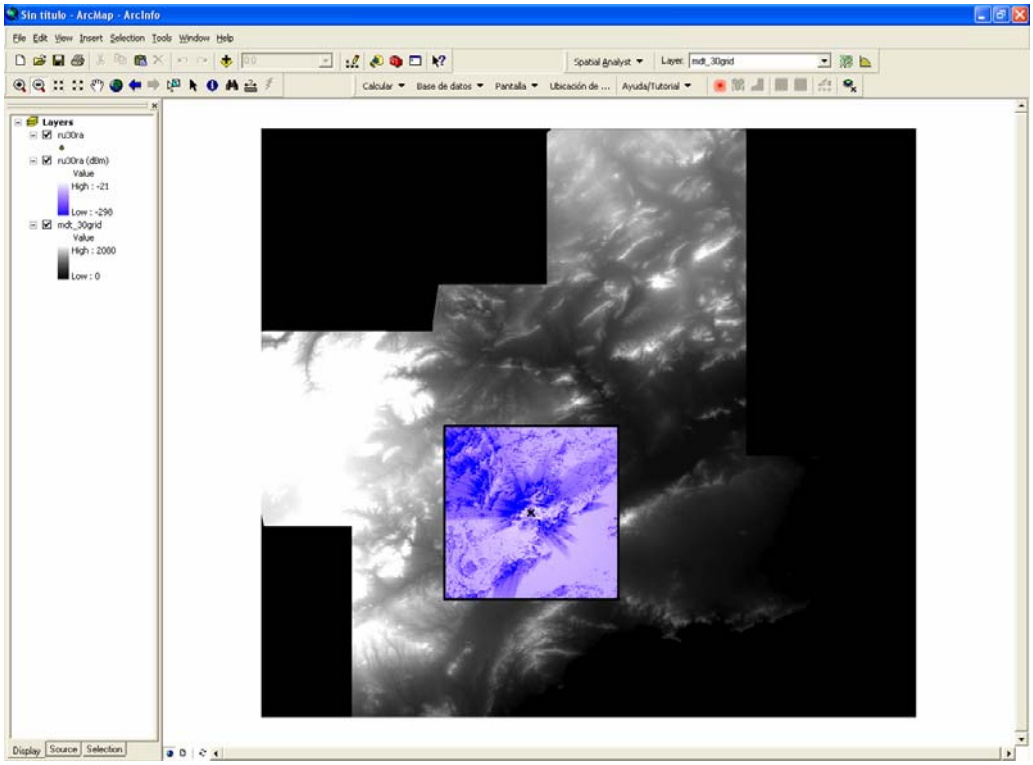


Figura 3.44. Pérdidas calculadas por el programa modificado sobre un área rectangular, para un MDT rural de resolución 30 metros.

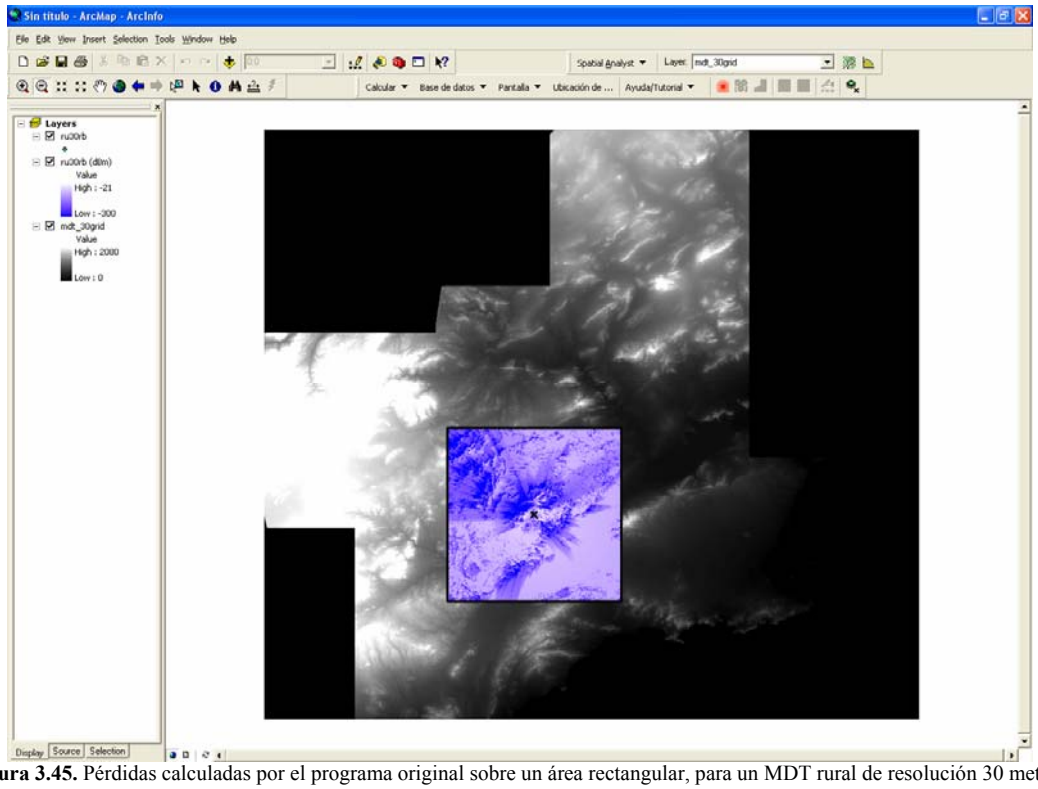


Figura 3.45. Pérdidas calculadas por el programa original sobre un área rectangular, para un MDT rural de resolución 30 metros.

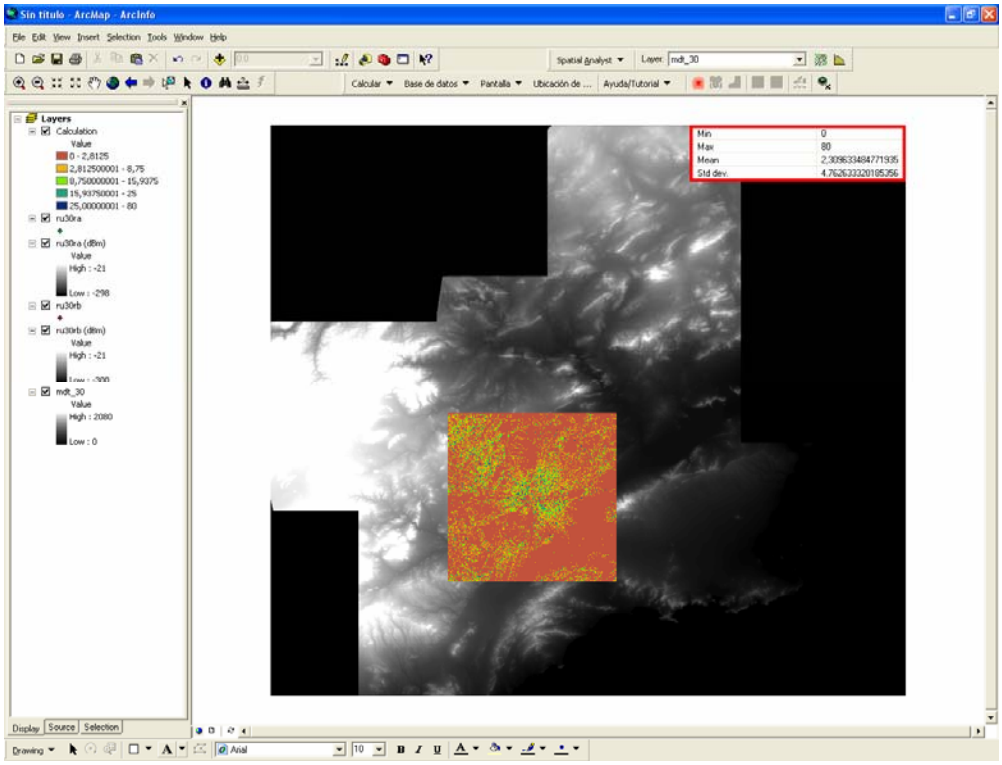


Figura 3.46. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT rural de resolución 30 metros.

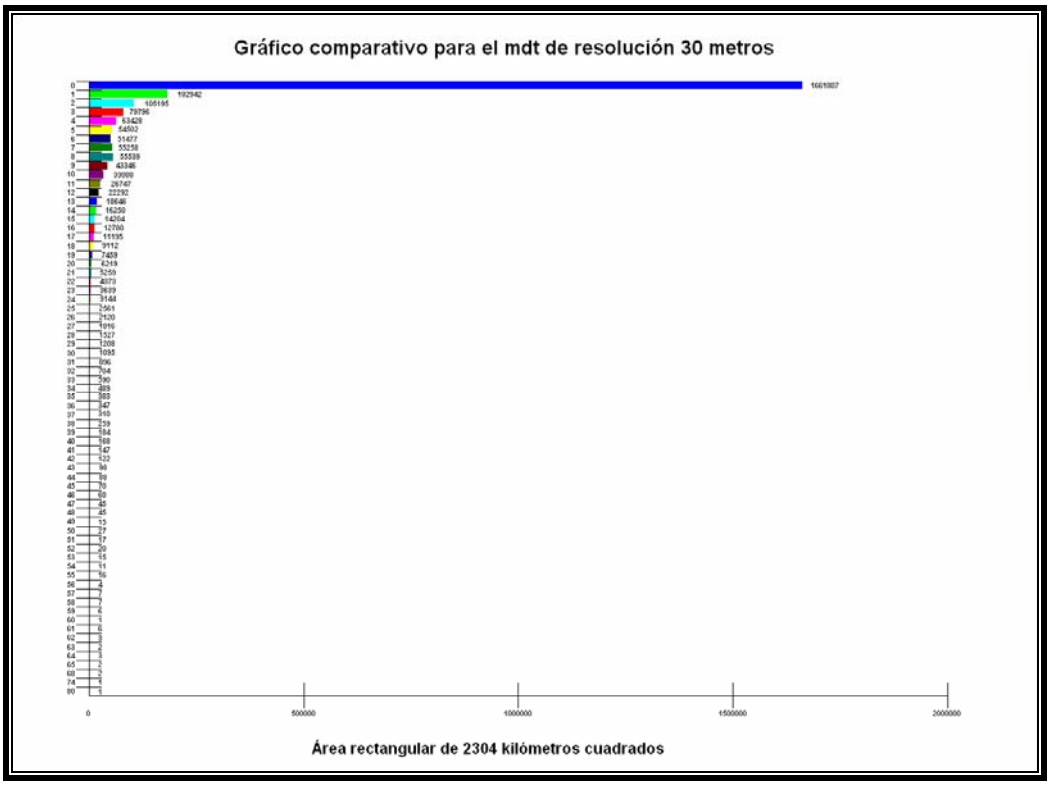


Figura 3.47. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT rural de resolución 30 metros.

3.5.4.2. Área circular.

MDT30GRID												
RADIO (Km)	ÁREA (Km^2)	TIEMPO										
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A		
						min	seg	min	seg	min	seg	
2,40	18,10	1,109	1,781	0,672	37,732	0	1,1	0	1,8	0	0,7	
4,80	72,38	11,719	16,734	5,015	29,969	0	11,7	0	16,7	0	5,0	
7,20	162,86	35,656	55,593	19,937	35,862	0	35,7	0	55,6	0	19,9	
9,60	289,53	76,860	125,844	48,984	38,924	1	16,9	2	5,8		49,0	
12,00	452,39	133,118	226,797	93,679	41,305	2	13,1	3	46,8	1	33,7	
14,40	651,44	198,813	360,218	161,405	44,808	3	18,8	6	0,2	2	41,4	
16,80	886,68	297,484	558,687	261,203	46,753	4	57,5	9	18,7	4	21,2	
19,20	1158,12	429,500	819,968	390,468	47,620	7	9,5	13	40,0	6	30,5	
21,60	1465,74	586,375	1144,625	558,250	48,771	9	46,4	19	4,6	9	18,3	
24,00	1809,56	792,297	1543,256	750,959	48,661	13	12,3	25	43,3	12	31,0	

Tabla 3.11. Tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 30 metros.

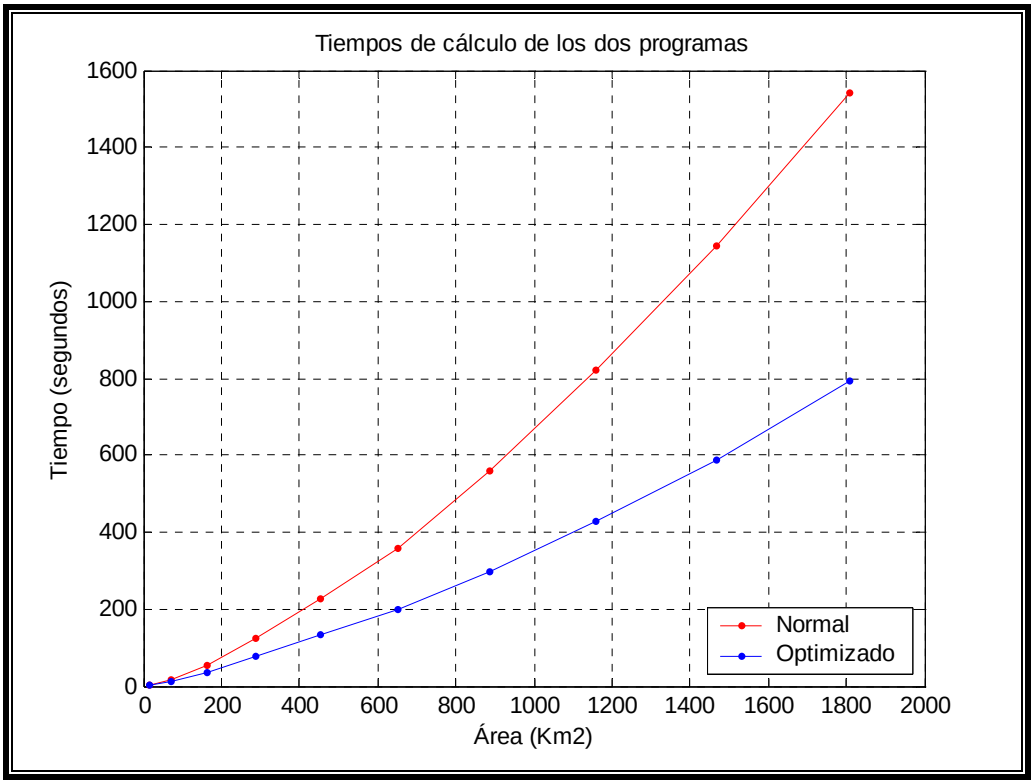


Figura 3.48. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 30 metros.

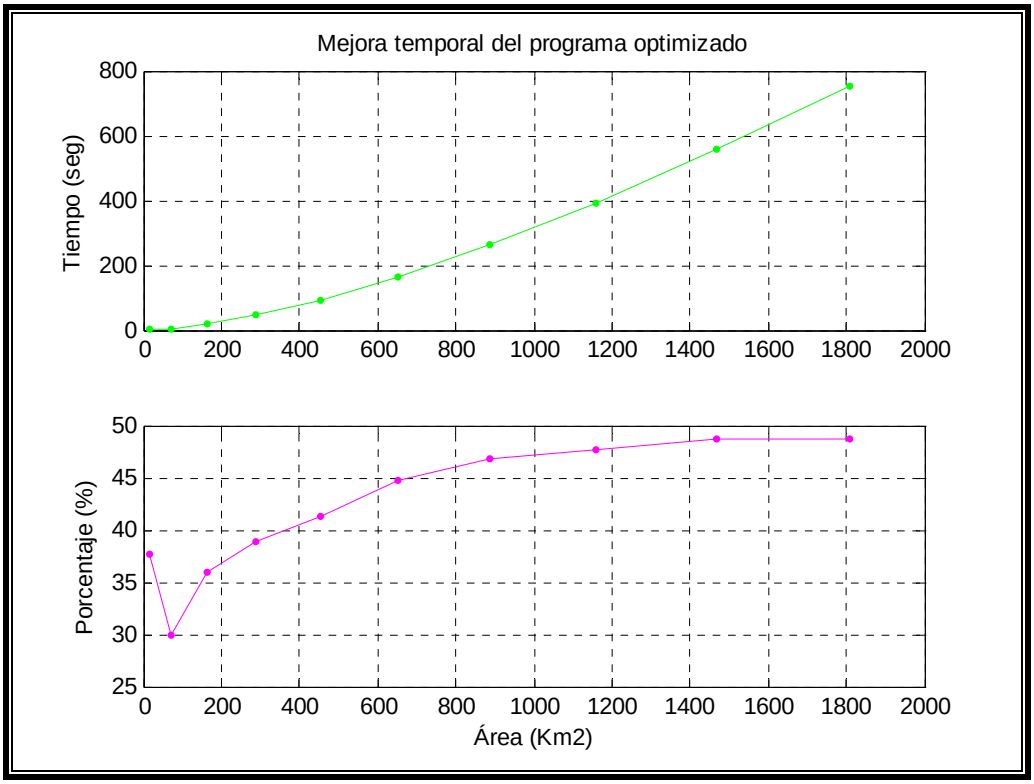


Figura 3.49. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT rural de resolución 30 metros.

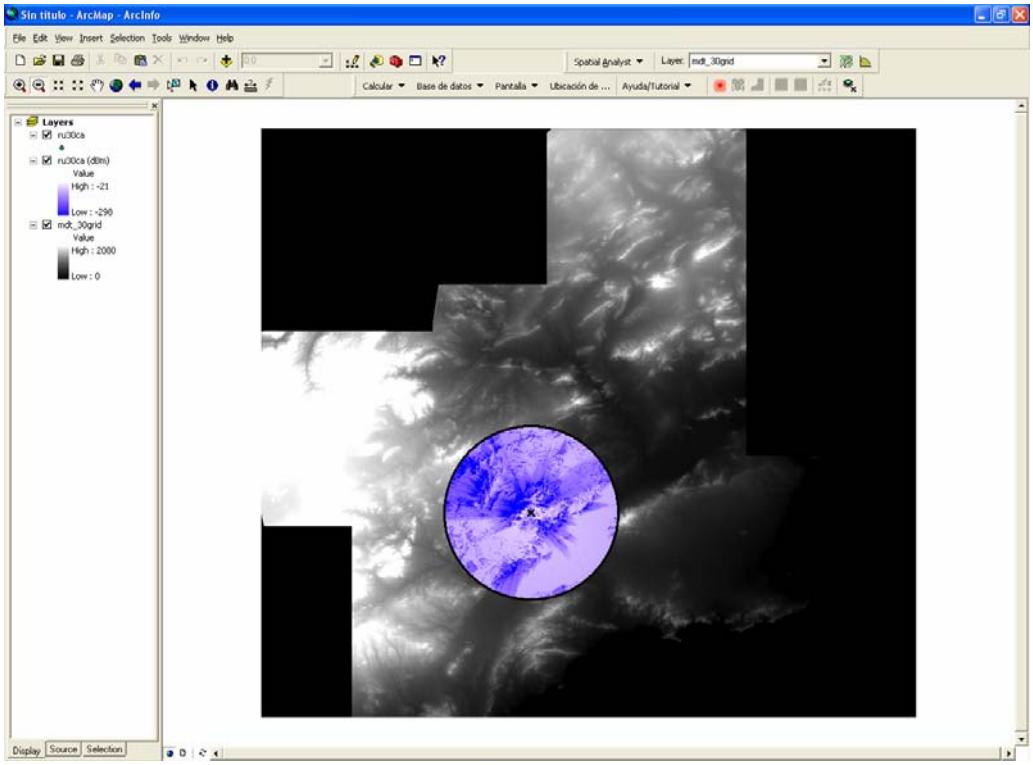


Figura 3.50. Pérdidas calculadas por el programa modificado sobre un área circular, para un MDT rural de resolución 30 metros.

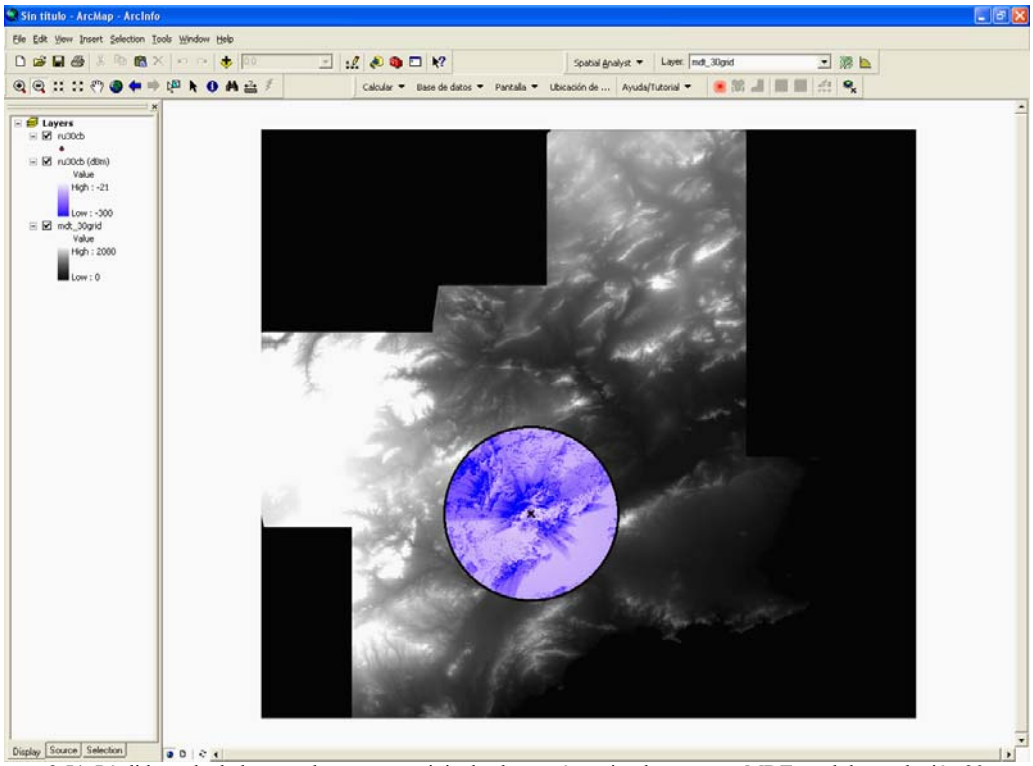


Figura 3.51. Pérdidas calculadas por el programa original sobre un área circular, para un MDT rural de resolución 30 metros.

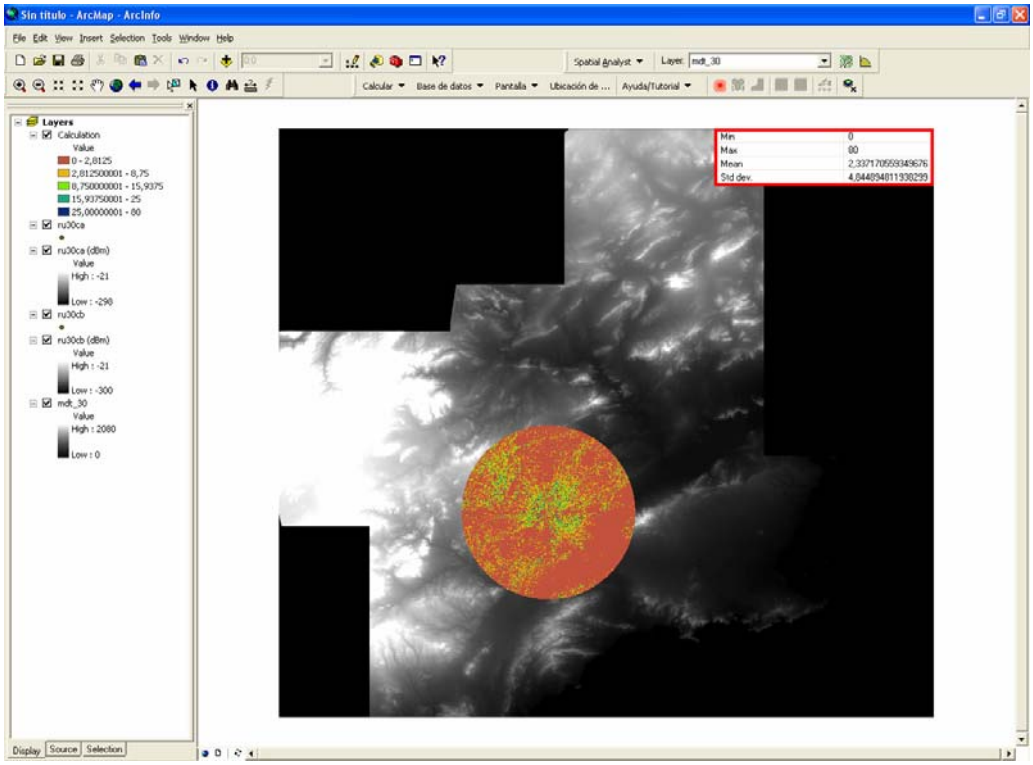


Figura 3.52. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT rural de resolución 30 metros.

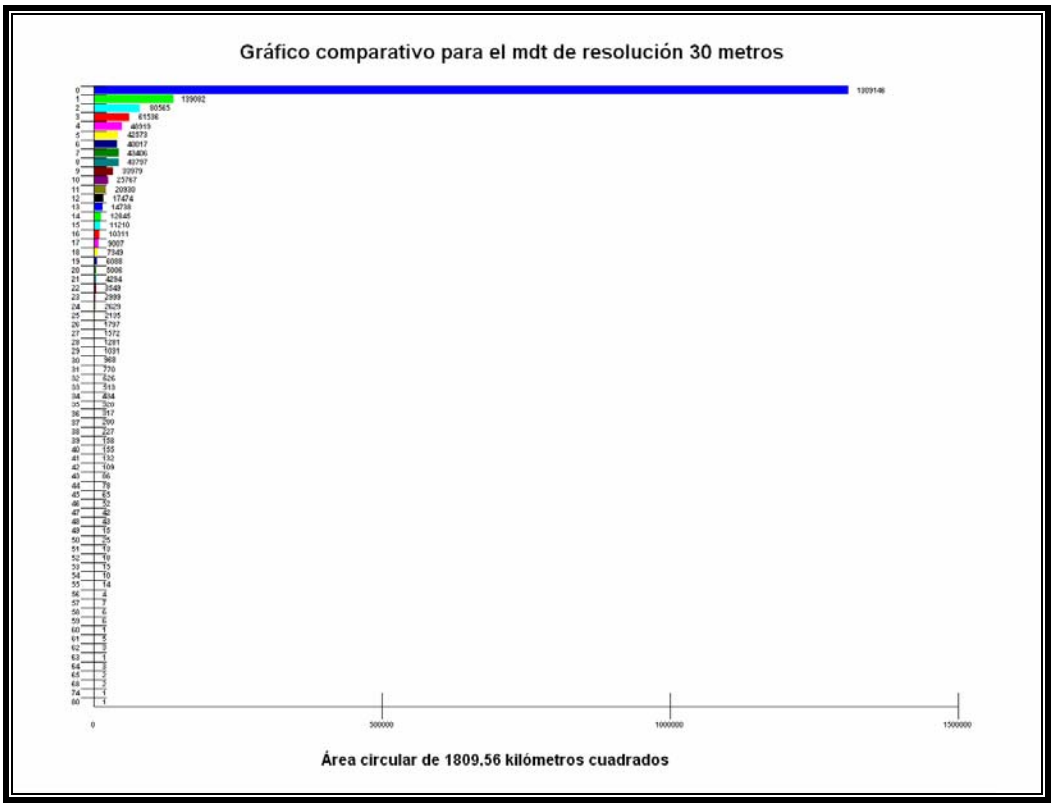


Figura 3.53. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT rural de resolución 30 metros.

ANÁLISIS DE TIEMPOS DE COMPUTACIÓN DEL MODELO URBANO

4. ANÁLISIS DE TIEMPOS DE COMPUTACIÓN DEL MODELO URBANO.

4.1. PROPAGACIÓN EN ENTORNOS URBANOS.

4.1.1. Introducción.

Cuando el terreno es orográficamente muy irregular o de tipo urbano, resulta difícil la modelización de obstáculos. Para la cobertura de estos escenarios de propagación se han ido desarrollando procedimientos empíricos de estimación de la pérdida básica de propagación y de la intensidad de campo. Todos ellos se basan en campañas de mediciones y en una posterior correlación de las medidas con características generales descriptivas del medio de propagación. *[3]*

Como los servicios de radiocomunicación de tipo zonal por antonomasia son los de radiodifusión y los móviles, estos han sido los primeros y destacados destinatarios de estos métodos empíricos de predicción. *[3]*

Los primeros métodos se presentaron en forma de ábacos y curvas de propagación normalizadas, para su utilización manual. Posteriormente se han ido desarrollando versiones y ampliaciones de los mismos, adaptadas al cálculo por ordenador, a fin de incorporarlas en programas informáticos. *[3]*

Los métodos empíricos proporcionan una estimación rápida de la pérdida básica de propagación o, alternativamente de la intensidad de campo en cualquier punto en torno a un transmisor. *[3]*

La señal recibida en los sistemas inalámbricos está afectada por las ***pérdidas en el espacio libre***, ***multitrayectorias*** y ***obstrucciones existentes entre el transmisor y el receptor***. Los modelos de propagación se usan ampliamente para predecir parámetros

como la dispersión del retardo y las pérdidas de propagación, importantes en el diseño y despliegue de sistemas de comunicaciones inalámbricas. *[3]*

Se van a presentar una serie de modelos para la predicción de pérdidas de propagación en entornos urbanos, recomendados por el IUT-R, los cuales son utilizados en *RADIOGIS*, y en consecuencia han sido modificados para lograr su optimización de velocidad de cálculo.

4.1.2. Modelo Xia-Bertoni.

El modelo Xia/Bertoni se aplica en entornos urbanos y suburbanos. El modelo es aplicable en caso de que el perfil sea como el indicado en la siguiente figura. *[4]*

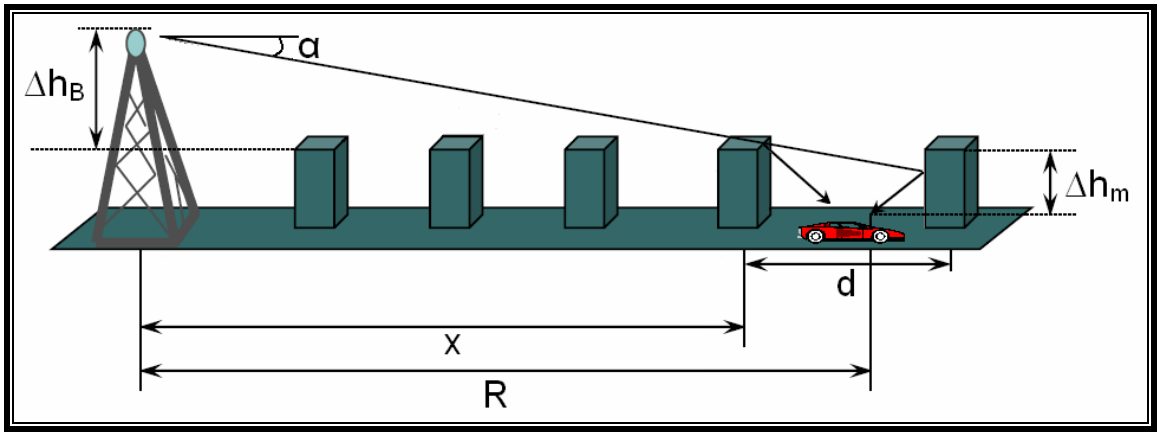


Figura 4.1. Modelo Xia-Bertoni. Parámetros significativos. *[6]*

Las pérdidas de propagación se expresan como la suma de los tres términos comentados anteriormente: perdidas por espacio libre, por la difracción desde la última difracción hasta el receptor y por la múltiple difracción sobre edificios. *[4]*

$$L = L_{bf} + L_{rts} + L_{md} \qquad (4.1)$$

Se toman tres casos en función de la altura de la antena de la estación base, y en función de cada uno, las pérdidas por propagación se calcularán de una determinada manera: *[4]*

1. Altura de la estación base cercana a la altura media de los edificios: *[4]*

$$L = -10*\log_{10}\left(\frac{\lambda}{2*\sqrt{2}*\pi*R}\right)^2 - 10*\log_{10}\left[\left(\frac{\lambda}{2*\pi^2*R}\right)*\left(\frac{1}{\theta}-\frac{1}{2*\pi+\theta}\right)^2\right] -$$

$$-10*\log_{10}\left(\frac{d}{R}\right)^2 \qquad (4.2)$$

Identificando parámetros: *[4]*

- $R \rightarrow$ distancia entre el transmisor y el receptor (km).
- $x \rightarrow$ distancia entre la antena transmisora y el edificio que produce la última difracción (m).
- $d \rightarrow$ distancia media de separación entre edificios (m).
- $\Delta h_m \rightarrow$ diferencia entre la altura media de los edificios y la altura de la antena transmisora (m).
- $r = \sqrt{(\Delta h_m)^2 + (x)^2}$. (4.3)
- $\theta = \tan^{-1}\left(\frac{\Delta h_m}{x}\right)$. (4.4)

2. Altura de la estación base por encima de la altura media de los edificios: *[4]*

$$L = -10*\log_{10}\left(\frac{\lambda}{4*\pi*R}\right)^2 - 10*\log_{10}\left[\left(\frac{\lambda}{2*\pi^2*R}\right)*\left(\frac{1}{\theta}-\frac{1}{2*\pi+\theta}\right)^2\right] -$$

$$-10*\log_{10}\left((2{,}35)^2*\left(\frac{\Delta h_B}{R}*\sqrt{\frac{d}{R}}\right)^{1{,}8}\right) \qquad (4.5)$$

Siendo Δh_B la altura de la antena de la estación base con respecto a la altura media de los edificios. *[4]*

3. Altura de la estación base por debajo de la altura media de los edificios: *[4]*

$$L = -10 * \log_{10}\left(\frac{\lambda}{2*\sqrt{2}*\pi*R}\right)^2 - 10*\log_{10}\left[\left(\frac{\lambda}{2*\pi^2*R}\right)*\left(\frac{1}{\theta}-\frac{1}{2*\pi+\theta}\right)^2\right] -$$

$$-10*\log_{10}\left\{\left[\frac{d}{2*\pi*(R-d)}\right]^2 * \frac{\lambda}{\sqrt{(\Delta h_B)^2+d^2}} * \left(\frac{1}{\phi}-\frac{1}{2*\pi+\phi}\right)^2\right\} \qquad (4.6)$$

Donde $\phi = -\tan^{-1}\left(\frac{\Delta h_B}{d}\right)$. (4.7)

Usando este modelo, la altura de la estación base puede estar situado por debajo de la altura media de los edificios. *[4]*

4.1.3. Modelo COST-231.

Para obtener una predicción más precisa, aparece el método COST 231, basado en los predecesores de Ikegami-Ioshida y el anteriormente comentado Xia-Bertoni, con la adaptación de algunas de sus variables a las características urbanísticas de las ciudades europeas. *[3]*

Este modelo fue desarrollado por el Grupo Europeo de trabajo COST-231, y se basa en la teoría de rayos con óptica geométrica y análisis de difracción, complementado con diferentes correcciones obtenidas a partir de campañas de medidas. *[3]*

Este modelo permite incorporar a la estimación Path Loss, más parámetros que describen las características de un ambiente urbano, como pueden ser las alturas de edificios, ancho de las calles, separación entre edificios y orientación del radioenlace entre el transmisor fijo y el receptor móvil. *[3]*

Las pérdidas en la propagación se producen debido a la difracción en los tejados de los edificios situados entre la estación base y el móvil, y a la dispersión en los edificios más próximos al móvil. *[3]*

El modelo supone que hay un rayo dominante (*AC*) que se propaga desde la antena transmisora hasta el edificio más próximo al receptor, por encima de los edificios situados entre ambos. Este rayo se difracta en el último edificio y origina dos componentes: una que llega directamente al receptor (*CB*), y otra que lo hace tras una reflexión en el edificio opuesto (*CDB*). *[3]*

Las ondas que llegan desde la estación base hasta el punto *C* sufren una pérdida por difracción debida a la proximidad entre el rayo *AC* y los edificios existentes entre A y *C*. Este conjunto de edificios se modela como pantallas difractantes separadas entre sí una distancia constante *b* igual a la separación media entre edificios. Así pues, al receptor llegan dos rayos: el *CB* que se difracta en *C*; y el *CDB* que se difracta en *C* y refleja en *D*. *[3]*

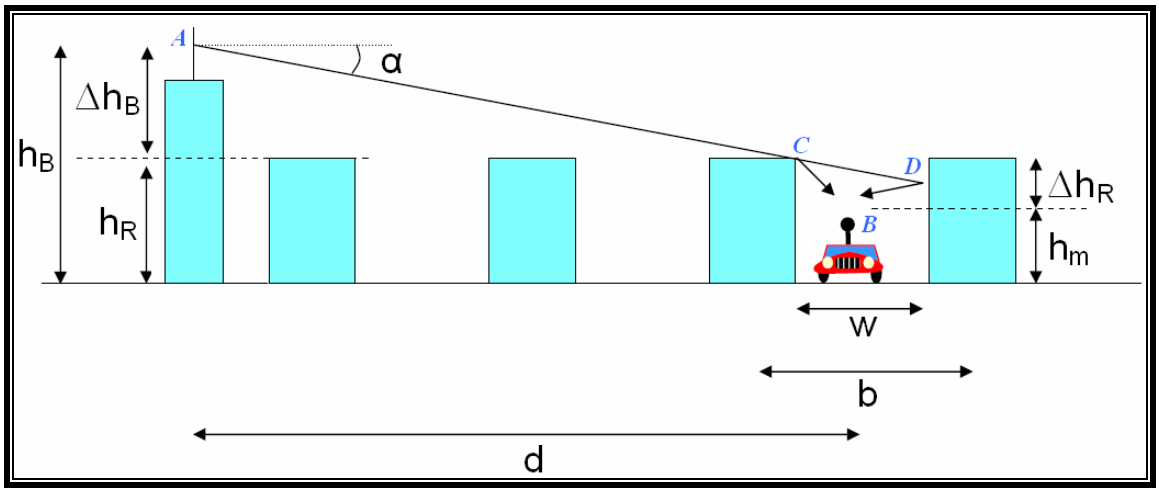


Figura 4.2. Modelo COST-231. Parámetros significativos. *[8]*

Este método está aceptado por la UIT-R en su recomendación 567-4, y presenta algunas restricciones que se deben cumplir para que el método sea aplicable en el cálculo de pérdidas correspondiente: *[3]*

- Banda de frecuencias entre 800 y 2000MHz.
- Altura del transmisor entre 4 y 50m.
- Altura del receptor entre 1 y 3m.
- Distancia entre el transmisor y el receptor desde 20m a 5km.

Los parámetros que intervienen en el método son los siguientes: *[3]*

- $h_B \rightarrow$ altura sobre el suelo de la antena de estación base fija (m).
- $h_m \rightarrow$ altura sobre el suelo de la antena del móvil (m).
- $h_R \rightarrow$ altura media de los edificios (m) ($h_R > h_m$).
- $w \rightarrow$ anchura de la calle donde se encuentra el móvil (m).
- $b \rightarrow$ distancia entre centros de edificios (m).
- $d \rightarrow$ distancia base-móvil (km).
- $\alpha \rightarrow$ ángulo de inclinación del rayo (º).
- $\varphi \rightarrow$ ángulo del rayo con el eje de la calle (º).
- $\Delta h_B = h_B - h_R \rightarrow$ altura de la antena de la estación base sobre la altura media de los edificios circundantes (m).
- $\Delta h_R = h_R - h_m \rightarrow$ altura de la antena de la estación base sobre la altura de antena del móvil (m). *[3]*

De acuerdo con el método, se distingue entre dos situaciones: cuando existe visión directa entre transmisor y receptor (caso LOS), y cuando no es así (caso NLOS). *[3]*

Para la primera de las situaciones, el modelo estima una fórmula simple para las pérdidas de propagación, diferente a la aplicada en el caso de espacio libre. Ésta se basa en mediciones llevadas a cabo en la ciudad de Stockholmo, y es la siguiente: *[3]*

$$L_b = 42{,}6 + 20 * \log_{10} f\,(\text{MHz}) + 26 * \log_{10} d \qquad (4.8)$$

En el supuesto de no existir visión directa, la pérdida básica de propagación para un radioenlace es: *[3]*

$$L_b = L_{bf} + L_{rts} + L_{msd} \qquad (4.9)$$

A continuación se explica el significado y cálculo de cada una de las pérdidas que conforman la pérdida total del radioenlace. *[3]*

1. L_{bf} es la pérdida en condiciones de espacio libre: *[3]*

$$L_{bf} = 32{,}45 + 20 * \log_{10} f\,(\text{MHz}) + 20 * \log_{10} d \qquad (4.10)$$

2. L_{rts} es la pérdida debida a la difracción entre el tejado de los edificios y el móvil: *[3]*

$$L_{rts} = -16{,}9 - 10 * \log_{10} w + 10 * \log_{10} f\,(\text{MHz}) + 20 * \log_{10} \Delta h_R + L_{ori} \qquad (4.11)$$

En caso que L_{rts} resulte igual o menor que 0, se considerará como pérdida 0dB´s. *[3]*

El valor de L_{ori} tiene en cuenta el ángulo entre el rayo y el eje de la calle (φ) de la manera que a continuación se muestra: *[3]*

$$L_{ori} = \begin{cases} -10 + 0{,}3571 * \varphi \rightarrow 0º < \varphi < 35º \\ 2{,}5 + 0{,}075 * (\varphi - 35º) \rightarrow 35º \leq \varphi \leq 55º \\ 4 - 0{,}114 * (\varphi - 55º) \rightarrow 55º \leq \varphi \leq 90º \end{cases} \qquad (4.11)$$

En la siguiente figura se puede observar este parámetro de orientación: *[3]*

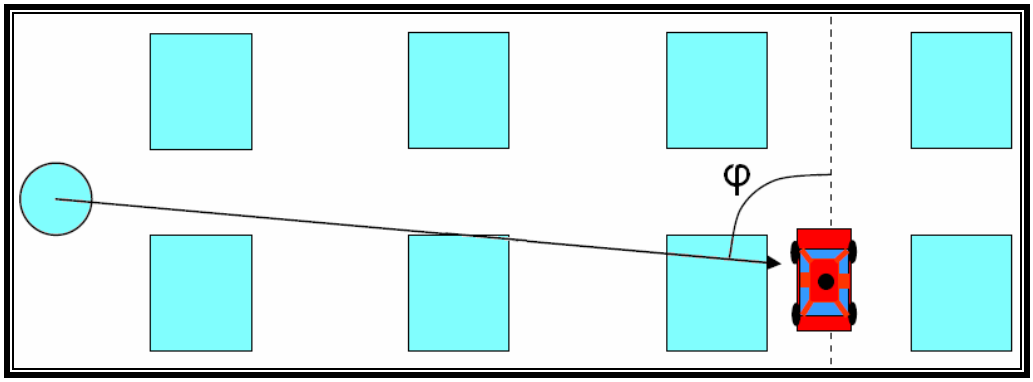


Figura 4.3. Modelo COST-231. Ángulo entre el rayo incidente y el eje de la calle en la que se encuentra el RX. *[8]*

3. L_{msd} estima las pérdidas debidas a la "difracción multiobstáculo" experimentada por el rayo entre la antena fija transmisora y el edificio próximo al receptor debido a los edificios interpuestos entre ambos: *[3]*

$$L_{msd} = L_{bsh} + k_a + k_d * \log_{10} d + k_f * \log_{10} f\,(\text{MHz}) - 9 * \log_{10} b \qquad (4.12)$$

Los distintos parámetros que forman parte de la expresión se calculan de la siguiente manera: *[3]*

$$L_{bsh} = -\,18 * \log_{10} (1+\Delta h_B);\ \text{si}\ \Delta h_B < 0 \rightarrow L_{bsh} = 0 \qquad (4.13)$$

$$k_a = \begin{cases} 54 \rightarrow \text{para}\ \Delta h_B \geq 0 \\ 54 - 0{,}8 * \Delta h_B \rightarrow \text{para}\ \Delta h_B < 0\ \text{y}\ d \geq 0{,}5 \\ 54 - 0{,}8 * \Delta h_B * d / 0{,}5 \rightarrow \text{para}\ \Delta h_B < 0\ \text{y}\ d < 0{,}5 \end{cases} \qquad (4.14)$$

$$k_d = \begin{cases} 18 \rightarrow \text{para}\ \Delta h_B \geq 0 \\ \\ 18 - 15 * \Delta h_B / h_r \rightarrow \text{para}\ \Delta h_B < 0 \end{cases} \qquad (4.15)$$

$$k_f = \begin{cases} -\,4 + 0{,}7 * (f / 925 - 1) \rightarrow \text{para ciudades de tamaño medio y zonas suburbanas} \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{con densidad de vegetación moderada.} \\ -\,4 + 1{,}5 * (f / 925 - 1) \rightarrow \text{para grandes centros metropolitanos.} \end{cases} \qquad (4.16)$$

4.2. CÓDIGO COMPLETO DEL MODELO URBANO ORIGINAL.

4.2.1. Fichero mapa.cpp.

```
#include <time.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <locale.h>

#define PI abs(acos(-1.0))
#define Re 8.5e6 /*Radio efectivo de la Tierra (m)*/

struct lconv *leng;
char sep,no_sep;


/***FUNCIONES DE LECTURA Y ESCRITURA***/

void lee_float (FILE*ent,float *pf){
   char s[40],*p;
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%f",pf);
}

void lee_double (FILE*ent,float *pf){
   char s[40],*p;
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%lf",pf);
}

void escribe_float (FILE*sal,float *pf){
   char s[40],*p;
   sprintf(s,"%.0f",*pf);
   while (p=strchr(s,no_sep))  *p=sep;
   fprintf(sal,"%s",s);
}

/*****************MAIN*************/

main (int argc,char *argv[]){
```

```
/***FUNCIONES PARA EL CÁLCULO DE LAS PÉRDIDAS POR PROPAGACIÓN SOBRE EDIFICIOS***/

float obp_wb(float v);


/***FUNCIONES PARA EL CÁLCULO DE LAS PÉRDIDAS POR DIFRACCIÓN FINAL***/

float dif_wb(float lambda,float r,float beta_final);


/***FICHEROS UTILIZADOS***/

FILE *grid; //rural+urbano
FILE *grid2; //urbano
FILE *fdatos;
FILE *resultados;
FILE *mdt; //rural + urbano
FILE *comprueba;
FILE *tiempo;


/***VARIABLES AUXILIARES***/

float pi = 3.141592654;
float G=0;
int i,j;
int fr,cr; /*Fila y columna del rx*/
long int prod,ind,prod1,ind1;
float *mapa,*campo,*mapa2;
char *tira;
char *tira2;
float temp;
int dx,dy,e1;
float e;
int *x,*y,*vec1;
int minx,maxx,miny,maxy;
float paso,aux;
float *perfil;
float *perfil2;
int cuadrante,posiciontx,fin,j_ini,n_nocero;
float hmax,hmin,hmed,H,hu;
int this_i,next_i,ultimo_edif;
float d; /*Distancia media entre edificios*/
float raux;
float q,F;
float ka,kd;
clock_t start, end;
```

```
/***PARAMETROS OBTENIDOS DEL FICHERO DE DATOS***/

float pire;
float frecuencia; /*MHz*/
float ht,hr,hrini,htini; /*Alturas de tx y rx*/
int nfils,ncols; /*Número de filas y columnas del grid*/
int ft,ct; /*Fila y columna donde se encuentra el tx*/
float celda; /*Lado de la celda en metros*/
char diag[80];
float azimut, elevacion;
int obstaculo;
float k;
float radio;
float figura,xllcorner,yllcorner;

int ncols2,nrows2,nodata2;
double xllcorner2,yllcorner2,cellsize2; /*Se guarda la cabecera del MDT genérico*/
int ncols1,nrows1,nodata1;
double xllcorner1,yllcorner1,cellsize1; /*Se guarda la cabecera del fichero de alturas*/
int ncols3,nrows3,nodata3;
double xllcorner3,yllcorner3,cellsize3; /*Se guarda la cabecera del fichero de alturas*/

int x1,y1;
int filasab,filasar,coli,cold;
int fa,fab,ci,cd;


/***VARIABLES UTILIZADAS COMO DATOS EN LOS CÁLCULOS FINALES***/

float R; /*Distancia tx-rx (en m)*/
float hvar; /*Rango de variación de las alturas de un perfil*/
int n_edif; /*Número de edificios en el perfil*/
float lambda,alfaultimo,alfapenult,t;
float r; /*Distancia 3D último edificio-rx(m)*/
float gamma_f; /*Ángulo de difracción final (rad)*/


/***VARIABLES PARA LOS RESULTADOS DE LOS CÁLCULOS FINALES***/

float theta, r_xia,, lam, phi;
float deltaHb, deltaHm, x_xia;
float Lfs, Lrts, Lmd;

float Lo,L_edif,L_dif,L;

start=clock();
leng=localeconv(); /*Pide el lenguaje actual*/
sep=*(leng->decimal_point);
if (sep==',')no_sep='.';
```

```
else no_sep=',';



/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO DE DATOS***/

frecuencia=atof(argv[7]);
htini=atof(argv[8]);
hrini=atof(argv[9]);
nfils=atoi(argv[10]);
ncols=atoi(argv[11]);
ct=atoi(argv[12]);
ft=atoi(argv[13]);
celda=atof(argv[14]);
obstaculo=atof(argv[15]);
k=atof(argv[16]);
azimut=atof(argv[17]);
elevacion=atof(argv[18]);

figura=atof(argv[20]);
radio=atof(argv[21]);
xllcorner=atof(argv[22]);
yllcorner=atof(argv[23]);



/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO DEL MDT GENÉRICO***/

if ((mdt=fopen(argv[5],"r"))!=NULL){

   if ((resultados=fopen(argv[3],"w"))!=NULL){

      tira=(char*)calloc(80,sizeof(char));

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&ncols2);
      fprintf(resultados,"%d\n",ncols2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&nrows2);
      fprintf(resultados,"%d\n",nrows2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&xllcorner2);
      fprintf(resultados,"%lf\n",xllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
```

```
      lee_double(mdt,&yllcorner2);
      fprintf(resultados,"%lf\n",yllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s      ",tira);
      lee_double(mdt,&cellsize2);
      fprintf(resultados,"%lf\n",cellsize2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s  ",tira);
      fscanf(mdt,"%d\n",&nodata2);
      fprintf(resultados,"%d\n",nodata2);

      fclose(resultados);
      free(tira); /*Libera la memoria reservada con calloc*/
   }

   else{
      printf("No se puede abrir fichero de salida\n");
      exit(1);
   }

}

lambda=300/frecuencia; /*Longitud de onda (m)*/


/***SE LEE LA CABECERA DEL FICHERO ALTURAS***/

if ((grid=fopen(argv[2],"r"))!=NULL){

   tira=(char*)calloc(80,sizeof(char));

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&ncols1);

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nrows1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&xllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&yllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&cellsize1);

   fscanf(grid,"%s\n",tira);
```

```
   fscanf(grid,"%d\n",&nodata1);

   free(tira); /*Libera la memoria reservada con calloc*/


   /***RESERVA DE MEMORIA PARA EL MDT (MAPA) Y EL MAPA DE CAMPO (CAMPO) RESULTADO***/
   prod=(long) nrows1*ncols1;
   mapa=(float*)calloc(prod,sizeof(float));
   campo=(float*)calloc(prod,sizeof(float));

   if (!mapa||!campo){
      printf("No queda memoria disponible para ciudad o campo\n");
      exit(1);
   }

   for (i=0;i<nrows1;i++)
   for (j=0;j<ncols1;j++){
      ind=(long) ncols1*i+j; /*Implementa una matriz con un vector*/
      lee_float(grid,&temp);
      if(fabs(temp+9999)<1)temp=0;
      mapa[ind]=temp; /*Almacena en mapa la información del MDT*/
   }

   fclose(grid);
}

else{
   printf("No se puede abrir fichero de entrada\n");
   exit(1);
}


/***SE LEE LA CABECERA DEL FICHERO ALTURAS***/

if ((grid2=fopen(argv[6],"r"))!=NULL){

   tira2=(char*)calloc(80,sizeof(char));

   fscanf(grid2,"%s\n",tira2);
   fscanf(grid2,"%d\n",&ncols3);

   fscanf(grid2,"%s\n",tira2);
   fscanf(grid2,"%d\n",&nrows3);

   fscanf(grid2,"%s\n",tira2);
   lee_double(grid2,&xllcorner3);

   fscanf(grid2,"%s\n",tira2);
   lee_double(grid2,&yllcorner3);
```

```
   fscanf(grid2,"%s\n",tira2);
   lee_double(grid2,&cellsize3);

   fscanf(grid2,"%s\n",tira2);
   fscanf(grid2,"%d\n",&nodata3);
   free(tira2); /*Libera la memoria reservada con calloc*/


   /***RESERVA DE MEMORIA PARA EL MDT (MAPA) Y EL MAPA DE CAMPO (CAMPO) RESULTADO***/

   prod=(long) nrows1*ncols1;
   mapa2=(float*)calloc(prod,sizeof(float));

   if (!mapa2){
      printf("No queda memoria disponible para ciudad o campo\n");
      exit(1);
   }

   for (i=0;i<nrows1;i++)
   for (j=0;j<ncols1;j++){
      ind=(long) ncols1*i+j; /*Implementar una matriz con un vector*/
      lee_float(grid2,&temp);
      if(fabs(temp+9999)<1)temp=0;
      mapa2[ind]=temp; /*Almacena en mapa la informacion del MDT*/
   }

   fclose(grid2);
}

else{
   printf("No se puede abrir fichero de entrada\n");
   exit(1);
}

fa=floor(((yllcorner1+(nrows1*cellsize1))-(yllcorner+(nfils*cellsize1)))/cellsize1);
fab=nrows1-fa-nfils;
ci=floor((xllcorner-xllcorner1)/cellsize1);
cd=ncols1-ci-ncols;


/***BUCLE PRINCIPAL***/

for (fr=0;fr<nrows1;fr++)
for (cr=0;cr<ncols1;cr++){
   ind=(long)ncols1*fr+cr;
   if ((mapa2[ind]==0) && (fr>fa-1) && (fr<fa+nfils) && (cr>ci-1) &&
   (cr<ci+ncols)){ /*Sólo se calcula en la zona seleccionada*/

      /***CÁLCULO DE LA DISTANCIA TX-RX***/
```

```
dx=cr-ct;
dy=fr-ft;
e=(float)sqrt(dx*dx+dy*dy); /*Distancia tx-rx (en numero de celdas)*/
R=e*(celda);
if ((figura==1) && (R > radio)) campo[ind]=-9999 ;
else{


   /***RESERVA DE MEMORIA***/

   e1=floor(e)+1; /*Tamaño de los vectores x e y*/
   x=(int*)calloc(e1,sizeof(int)); /*En x[] se guardan las columnas y en y[] las
                                    filas del gris que determinan los puntos
                                    pertenecientes al perfil*/
   y=(int*)calloc(e1,sizeof(int));
   if(!x||!y){
      printf("No queda memoria disponible para coordenadas\n");
      exit(1);
   }


   /***CREACIÓN DEL VECTOR X***/

   paso=(e==0) ? 0 : dx/floor(e);
   for (i=0;i<(e1-1);i++){
      aux=ct+i*paso-floor(ct+i*paso);
      if (aux<0.5) x[i]=floor(ct+i*paso);
      else x[i]=ceil(ct+i*paso);
   }
   x[e1-1]=cr; /*El último punto no se interpola, al igual que el primero*/


   /***CREACIÓN DEL VECTOR Y***/

   paso=(e==0) ? 0 : dy/floor(e);
   for (i=0;i<(e1-1);i++){
      aux=ft+i*paso-floor(ft+i*paso);
      if (aux<0.5) y[i]=floor(ft+i*paso);
      else y[i]=ceil(ft+i*paso);
   }

   y[e1-1]=fr; /*Los elementos de los vectores x e y están ordenados de menor a
                mayor*/


   /***SE HALLA EL PERFIL***/

   perfil=(float*)calloc(e1,sizeof(float));
   perfil2=(float*)calloc(e1,sizeof(float));
```

```
if (!perfil){
   printf("No queda memoria disponible para perfil\n");
   exit(1);
}

if (!perfil2){
   printf("No queda memoria disponible para perfil\n");
   exit(1);
}

for(j=0;j<e1;j++){
   ind=(long) ncols1*y[j]+x[j];
   perfil2[j]=mapa2[ind];
   perfil[j]=mapa[ind];
}

free(x);
free(y);

hr = hrini + perfil[e1-1];
ht = htini + perfil[0] - perfil2[0];


/***SE DETERMINA SI EL TX ESTÁ EN LA CALLE O EN UN EDIFICIO***/
if (perfil2[0]==0) posiciontx=0; /*Calle*/
else posiciontx=1; /*Edificio*/


/***ALMACENAMIENTO DE LOS ÍNDICES DEL PERFIL EN LOS QUE HAY EDIFICIOS***/

vec1=(int*)  calloc(e1,sizeof(int));  /*Si  el  tx  esta  en  un  edificio,  este
                                        edificio no cuenta*/
if (!vec1){
   printf("No queda memoria disponible para vec1\n");
   exit(1);
}

j_ini=fin=0;
if (posiciontx==1)
do{
   if (perfil2[j_ini++]==0) fin=1;
}

while (fin==0);
hmed=hu=0;
for (j=j_ini,n_nocero=0;j<e1;j++){
   if (perfil2[j]>0){
      if (n_nocero==0) hmax=hmin=perfil[j];
```

```
            vec1[n_nocero++]=j; /*Se guarda en n_nocero el numero de componentes de
                                  vec1*/
            hmed+=perfil[j];
            hu=perfil[j];
            if (hu<hmin) hmin=hu;
            if (hmax<hu) hmax=hu;
         }

      }

      if (n_nocero==0) goto vision_directa;
      else hmed=hmed/n_nocero;
      H=ht-hmed; /*Altura del tx sobre los edificios*/


      /*CÁLCULO DE LA DISTANCIA MEDIA ENTRE EDIFICIOS*/

      d=0;
      n_edif=ultimo_edif=0;
      for (j=0;j<(n_nocero-2);j++){
         this_i=vec1[j];
         next_i=vec1[j+1];
         if (perfil2[this_i]>perfil2[next_i]){
            n_edif+=1;
            if (ultimo_edif!=0) d+=(this_i-ultimo_edif)*celda;
            ultimo_edif=this_i;
         }

         else if ((next_i-this_i)>1){
            n_edif+=1;
            if (ultimo_edif!=0) d+=(this_i-ultimo_edif)*celda;
            ultimo_edif=this_i;
         }

      }

      if (ultimo_edif!=0) d+=(vec1[n_nocero-1]-ultimo_edif)*celda; /*n_edif=número de
                                                edificios en el perfil sin contar el
                                                edificio del tx, ni el último
                                                edificio*/
      d=(n_edif>0)? d/n_edif : 0; /*Si n_edif==0 no habrá atenuación por propagación
                                    sobre edificios*/


      /***DISTANCIA 2D ÚLTIMO EDIFICIO-RX (M)***/

      raux=((e1-1)-vec1[n_nocero-1])*celda;
```

```
        /***ÁNGULOS DE INCIDENCIA DEL RAYO SOBRE EL ÚLTIMO Y PENÚLTIMO EDIFICIO (RAD)
        TENIENDO EN CUENTA LA CURVATURA DE LA TIERRA***/

        alfaultimo=atan(((ht-hu)/R)-(R/(2*Re)));
        alfapenult=atan((H/R)-(R/(2*Re)));


        /***PARÁMETRO T***/

        t=alfapenult*sqrt(d/lambda)*sqrt(PI);


        /***DISTANCIA 3D ÚLTIMO EDIFICIO-RX (M)***/

        r=sqrt(pow(raux,2)+pow(hu-hr,2));


        /***ÁNGULO DE DIFRACCIÓN FINAL (RAD)***/

        gamma_f=(raux!=0) ? atan((hmed-ht)/raux) : atan ((hmed-hr)/celda);

        free(vec1);
        free(perfil);
        free(perfil2);


        /***CÁLCULO DE LAS PÉRDIDAS (DB)***/


////////***ESPACIO LIBRE***/////

        if (R>0) Lo=-20*log10(lambda/(4*PI*R));
        else Lo=0.0;
////////////////////////////


////////***MODELO DE WALFISCH-BERTONI***/////

        if (!strcmp(argv[4],"wb")){
           if (n_edif>0){
              q=obp_wb(t/sqrt(PI));
              if (q>0) L_edif=-20*log10(q)+0.525*hvar;
              else L_edif=0; /*El modelo no es aplicable bajo estas condiciones*/
           }

           else L_edif=0;
           if (hmed < hr)
           F = 1;
           else{
```

```
                F=dif_wb(lambda,r,gamma_f-alfaultimo);
                F=F*sqrt(2); /*Efecto multipath*/
            }

            L_dif=-20*log10(F);
         }
//////////////////////////////////////


/////***MODELO COST-231***/////

         if (! strcmp(argv[4],"cost")){
            if (n_edif>0) {
               if (H<0) {
                  if (R/1000<0.5) ka = 54 - 0.8*H*(R/1000)/0.5;
                  else ka = 54 - 0.8*H;
               }

               else ka = 54;
               if ((H<0)&&(hmed!=hr)) kd = 18 - 15*H/(hmed - hr);
               else kd = 18;
               if (fabs(1 + H)==0)
               L_edif = -18*log10 (0.0000000001) + ka + kd*log10(R/1000) –
               (4 + 0.7*(frecuencia/925 - 1))*log10(frecuencia) - 9*log10(d);

               else
               L_edif = -18*log10 (fabs(1 + H)) + ka + kd*log10(R/1000) –
               (4 + 0.7*(frecuencia/925 - 1))*log10(frecuencia) - 9*log10(d);

               if (hmed <= hr)
               L_dif = 0;
               else
               L_dif =  -16.9 -10*log10(raux+0.00001) + 10*log10(frecuencia)
               + 20*log10(fabs(hmed - hr));

            }

            else {
               L_edif = 0;
               if (hmed <= hr)
               L_dif = 0;
               else
               L_dif =  -16.9 -10*log10(raux+0.00001) + 10*log10(frecuencia)
               + 20*log10(fabs(hmed - hr));

            }

         }
//////////////////////////////////////
```

```
/////***MODELO XIA-BERTONI***/////

        if (! strcmp(argv[4],"xia")){

     x_xia=raux+0.00001;
     lam = 3e8/(frecuencia*1e6);
     deltaHb=ht-hmed;
     deltaHm=hmed-hr;

     if (n_edif>0) {
        theta = atan(deltaHm/x_xia);
        if (theta<=0) theta = 0.02;
        r_xia = sqrt(pow(deltaHm,2)+pow(x_xia,2));
        phi=-atan(deltaHb/d);


  /***TRES SUPUESTOS PARA EL MODELO XIA-BERTONI***/

  ///CASO 1: ANTENAS TRANSMISORAS CERCA DEL NIVEL EDIO DE LOS EDIFICIOS

          if((ht>=(hmed-0.5))&&(ht<=(hmed+0.5))){ /*Se pone un margen de 1 m de
                                                  diferencia entre hmed y ht*/
             Lfs = -10*log10(pow((lam/(2*sqrt(2)*PI*R)),2));
             if (hmed < hr)
             Lrts = 0;
             else
             Lrts=-10*log10((lam/(2*pow(PI,2)*r_xia))*pow(((1/theta)-1/(2*PI+theta))
             ,2));
             Lmd = -10*log10(pow(d/R,2));
          }

  ///CASO 2: ANTENAS TRANSMISORAS POR ENCIMA DEL NIVEL EDIO DE LOS EDIFICIOS

          else if(ht>hmed){
             Lfs = -10*log10(pow((lam/(4*PI*R)),2));
             if (hmed < hr)
             Lrts = 0;
             else
             Lrts=-10*log10((lam/(2*pow(PI,2)*r_xia))*pow(((1/theta)-1/(2*PI+theta))
             ,2));
             Lmd = -10*log10(pow(2.35,2)*pow(((deltaHb/R)*sqrt(d/lam)),1.8));
          }

  ///CASO 3: ANTENAS TRANSMISORAS POR DEBAJO DEL NIVEL EDIO DE LOS EDIFICIOS

          else if(ht<hmed){
             Lfs = -10*log10(pow((lam/(2*sqrt(2)*PI*R)),2));
             if (hmed < hr)
             Lrts = 0;
```

```
                else
                Lrts=-10*log10((lam/(2*pow(PI,2)*r_xia))*pow(((1/theta)-1/(2*PI+theta))
                ,2));
                Lmd=-10*log10(pow((d/(2*PI*(R-d))),2)*(lam/sqrt(pow(deltaHb,2)+pow(d,2)))
                * pow((1/phi-1/(2*PI+phi)),2));
             }

          }

          else { /*Si el número de edificios es 0, no hay pérdidas por difracción
                 multipantalla (Lmd)*/
          Lmd=Lrts=0;

          if (x_xia>0) {
             theta = atan(deltaHm/x_xia);
             if (theta<=0) theta = 0.02;
             r_xia = sqrt(pow(deltaHm,2)+pow(x_xia,2))+0.00001;
             if (hmed < hr)
             Lrts = 0;
             else
             Lrts=-10*log10((lam/(2*pow(PI,2)*r_xia))*pow(((1/theta)- 1/(2*PI+theta))
             ,2));
          }

   ///CASO 1: ANTENAS TRANSMISORAS CERCA DEL NIVEL EDIO DE LOS EDIFICIOS

.........if((ht>=(hmed-0.5))&&(ht<=(hmed+0.5))){ /*Se pone un margen de 1 m de
                                                   diferencia entre hmed y ht*/
             Lfs = -10*log10(pow((lam/(2*sqrt(2)*PI*R)),2));
          }

   ///CASO 2: ANTENAS TRANSMISORAS POR ENCIMA DEL NIVEL EDIO DE LOS EDIFICIOS

          else if (ht>hmed) {
             Lfs = -10*log10(pow((lam/(4*PI*R)),2));
          }

   ///CASO 3: ANTENAS TRANSMISORAS POR DEBAJO DEL NIVEL EDIO DE LOS EDIFICIOS

          else if (ht<hmed) {
             Lfs = -10*log10(pow((lam/(2*sqrt(2)*PI*R)),2));
          }

       }

       L = Lfs + Lrts + Lmd;

       goto fin_bucle;
   }
////////////////////////////////////////
```

```
         L=Lo+L_edif+L_dif;
         goto fin_bucle;


         /***VISIÓN DIRECTA***/

         vision_directa:

         if (R>0){
            R=sqrt(R*R+(ht-hr)*(ht-hr));
            L=-20*log10(lambda/(4*PI*R));
         }

         else L=0;
         fin_bucle:

         campo[ind]= L-G;
      }

      free (perfil);
      free (perfil2);
      free (vec1);
   }

   else campo[ind]=-9999;
} /***FIN DEL BUCLE PRINCIPAL***/


/***ESCRITURA DEL FICHERO DE CAMPO EN DISCO***/

if ((resultados=fopen(argv[3],"a"))!=NULL){

   filasar=((yllcorner2+cellsize2*nrows2)-(yllcorner1+cellsize1*nrows1))/cellsize2;
   filasab=nrows2-nrows1-filasar;
   coli=(xllcorner1-xllcorner2)/cellsize2;
   cold=ncols2-ncols1-coli;

   for (x1=0;x1<filasar;x1++){
      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   for (i=0;i<nrows1;i++){
      for (y1=0;y1<coli;y1++){
         fprintf(resultados,"-9999");
```

```
         fprintf(resultados," ");
      }

      for (j=0;j<ncols1;j++){
         ind1=(long) ncols1*i+j;
         escribe_float(resultados,&(campo[ind1]));
         fprintf(resultados," ");
      }

      for (y1=0;y1<cold;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   for (x1=0;x1<filasab;x1++){
      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   fclose (resultados);
   printf("El campo está  en disco\n");
}

else{
   printf("No se puede abrir fichero de salida\n");
   exit (1);
}

free(campo);
free(mapa);
free (mapa2);

comprueba=fopen(argv[24],"w");
fprintf(comprueba,"\n");

fclose (comprueba);
end = clock();

if  ((tiempo=fopen("c:\\arcgis\\RAGIS\\tiempo.txt","w"))!=NULL){  /*Abre  el  fichero  de
                                                                   campo para escritura*/
   fprintf(tiempo,"The time was: %f\n", (end - start) / CLK_TCK);
   fclose(tiempo);
```

```
}

} /***FIN DE MAIN***/
```

4.2.2. Fichero opb_wb.cpp.

```
#include <math.h>

float obp_wb (float v) {

   float q;

   q=2.34*pow(v,0.9);
   return q;
}
```

4.2.3. Fichero dif_wb.cpp.

```
#include <math.h>

#define PI fabs(acos(-1.))

float dif_wb (float lambda,float r,float beta) {

   float F;

   F=sqrt(lambda)/(2*PI*sqrt(r));
   F=F*fabs((1/beta)-(1/(2*PI+beta)));
   return F;
}
```

4.3. CÓDIGO COMPLETO DEL MODELO URBANO MODIFICADO.

De forma similar al apartado 1.3., segidamente se muestra el código diseñado a partir del proyecto primario "urbano.ide", destacando con comentarios numerados del 1 al 33, todos y cada uno de los cambios realizados en él.

4.3.1. Fichero mapa.cpp.

```
#include <time.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <locale.h>

#define PI abs(acos(-1.0))

struct lconv *leng;
char sep,no_sep;


/***FUNCIONES DE LECTURA Y ESCRITURA***/

void lee_float (FILE*ent,float *pf){
   char s[40],*p;
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%f",pf);
}

void lee_double (FILE*ent,float *pf){
   char s[40],*p;
   fscanf(ent,"%s",s);
   while (p=strchr(s,no_sep))  *p=sep;
   sscanf(s,"%lf",pf);
}

void escribe_float (FILE*sal,float *pf){
   char s[40],*p;
   sprintf(s,"%.0f",*pf);
   while (p=strchr(s,no_sep))  *p=sep;
   fprintf(sal,"%s",s);
```

```
}


/*****************MAIN*************/

main (int argc,char *argv[]){

float CalculaAtenuacion(float *perfil, float *perfil2, float hrini, float htini, int e1,
float celda, int eleccion, float frecuencia, int ep, float R, int *x, int *y, int ft,
int ct, float *mapa2, int *ind_calc, float figura, float radio );
/*1. Se declara el método CalculaAtenuacion y se elimina la defición de los métodos
obp_wb y dif_wb*/


/***FICHEROS UTILIZADOS***/

FILE *grid;
FILE *grid2;
FILE *fdatos;
FILE *resultados;
FILE *mdt;
FILE *comprueba;
FILE *tiempo;


/***VARIABLES AUXILIARES***/

float pi = 3.141592654;
float G=0;
int i,j;
int fr,cr;
long int prod,ind,prod1,ind1;
float *mapa,*campo,*mapa2;
char *tira;
char *tira2;
float temp;
int dx,dy,e1;
float e;
int *x,*y;
int minx,maxx,miny,maxy;
float paso,aux;
float *perfil;
float *perfil2;
int cuadrante;
int *ind_calc;
int ep;
clock_t start, end;
```

```
/*2. Se eliminan los parámetros que se definirán en el método CalculaAtenuacion ('*vec1', 'posiciontx', 'fin', 'j_ini', 'n_nocero', 'hmax', 'hmin', 'hmed', 'H', 'hu', 'this_i', 'next_i', 'ultimo_edif', 'd', 'raux', 'q', 'F', 'ka', 'kd'), y se declaran las nuevas variables 'ind_calc' y 'ep'*/


/***PARAMETROS OBTENIDOS DEL FICHERO DE DATOS***/

float pire;
float frecuencia;
float hrini,htini;
int nfils,ncols;
int ft,ct;
float celda;
char diag[80];
float azimut, elevacion;
int obstaculo;
float k;
float radio;
float figura,xllcorner,yllcorner;
int eleccion;
/*3. Se eliminan los parámetros que se definirán en el método CalculaAtenuacion ('ht', 'hr'), y se declara la nueva variable 'eleccion'*/

int ncols2,nrows2,nodata2;
double xllcorner2,yllcorner2,cellsize2;
int ncols1,nrows1,nodata1;
double xllcorner1,yllcorner1,cellsize1;
int ncols3,nrows3,nodata3;
double xllcorner3,yllcorner3,cellsize3;

int x1,y1;
int filasab,filasar,coli,cold;
int fa,fab,ci,cd;


/***VARIABLES UTILIZADAS COMO DATOS EN LOS CÁLCULOS FINALES***/

float R;
float L;
/*4. Se eliminan los parámetros que se definirán en el método CalculaAtenuacion ('hvar', 'n_edif', 'lambda', 'alfaultimo', 'alfapenult', 't', 'r', 'gamma_f', 'l_edif', 'l_dif', 'Lo'*/

start=clock();
leng=localeconv();
sep=*(leng->decimal_point);
if (sep==',')no_sep='.';
else no_sep=',';
```

```
/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO DE DATOS***/

frecuencia=atof(argv[7]);
htini=atof(argv[8]);
hrini=atof(argv[9]);
nfils=atoi(argv[10]);
ncols=atoi(argv[11]);
ct=atoi(argv[12]);
ft=atoi(argv[13]);
celda=atof(argv[14]);
obstaculo=atof(argv[15]);
k=atof(argv[16]);
azimut=atof(argv[17]);
elevacion=atof(argv[18]);

figura=atof(argv[20]);
radio=atof(argv[21]);
xllcorner=atof(argv[22]);
yllcorner=atof(argv[23]);

if (!strcmp(argv[4],"wb")){
   eleccion=1;
}
/*5. La variable 'eleccion' será igual a 1 si en el entorno gráfico ArcMap se elige que
el cálculo de pérdidas sea realizado con el método Walfisch-Bertoni*/

else if (! strcmp(argv[4],"cost")){
   eleccion=2;
}
/*6. La variable 'eleccion' será igual a 2 si en el entorno gráfico ArcMap se elige que
el cálculo de pérdidas sea realizado con el método COST-231*/

if (!strcmp(argv[4],"xia")){
   eleccion=3;
}
/*7. La variable 'eleccion' será igual a 3 si en el entorno gráfico ArcMap se elige que
el cálculo de pérdidas sea realizado con el método Xia-Bertoni*/


/***OBTENCIÓN DE LOS PARÁMETROS DEL FICHERO DEL MDT GENÉRICO***/

if ((mdt=fopen(argv[5],"r"))!=NULL){

   if ((resultados=fopen(argv[3],"w"))!=NULL){

      tira=(char*)calloc(80,sizeof(char));

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
```

```
      fscanf(mdt,"%d\n",&ncols2);
      fprintf(resultados,"%d\n",ncols2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s         ",tira);
      fscanf(mdt,"%d\n",&nrows2);
      fprintf(resultados,"%d\n",nrows2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&xllcorner2);
      fprintf(resultados,"%lf\n",xllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s     ",tira);
      lee_double(mdt,&yllcorner2);
      fprintf(resultados,"%lf\n",yllcorner2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s      ",tira);
      lee_double(mdt,&cellsize2);
      fprintf(resultados,"%lf\n",cellsize2);

      fscanf(mdt,"%s\n",tira);
      fprintf(resultados,"%s  ",tira);
      fscanf(mdt,"%d\n",&nodata2);
      fprintf(resultados,"%d\n",nodata2);

      fclose(resultados);
      free(tira);
   }

   else{
      printf("No se puede abrir fichero de salida\n");
      exit(1);
   }

}
/*8.  Se  elimina  la  inicialización  de  'lambda'  (se  hará  al  principio  de
CalculaAtenuacion) */


/***SE LEE LA CABECERA DEL FICHERO ALTURAS***/

if ((grid=fopen(argv[2],"r"))!=NULL){

   tira=(char*)calloc(80,sizeof(char));

   fscanf(grid,"%s\n",tira);
```

```
   fscanf(grid,"%d\n",&ncols1);

   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nrows1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&xllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&yllcorner1);

   fscanf(grid,"%s\n",tira);
   lee_double(grid,&cellsize1);
   fscanf(grid,"%s\n",tira);
   fscanf(grid,"%d\n",&nodata1);

   free(tira);


   /***RESERVA DE MEMORIA PARA EL MDT (MAPA) Y EL MAPA DE CAMPO (CAMPO) RESULTADO***/

   prod=(long) nrows1*ncols1;
   mapa=(float*)calloc(prod,sizeof(float));
   campo=(float*)calloc(prod,sizeof(float));

   if (!mapa||!campo){
      printf("No queda memoria disponible para ciudad o campo\n");
      exit(1);
   }

   fa=floor(((yllcorner1+(nrows1*cellsize1))-(yllcorner+(nfils*cellsize1)))/cellsize1);
   fab=nrows1-fa-nfils;
   ci=floor((xllcorner-xllcorner1)/cellsize1);
   cd=ncols1-ci-ncols;
/*9. Estas cuatro variables se calculan antes que en el primario para poder realizar la
comprobación siguiente (modificación número 9) */

   for (i=0;i<nrows1;i++)
   for (j=0;j<ncols1;j++){
      ind=(long) ncols1*i+j;
      lee_float(grid,&temp);
      if(fabs(temp+9999)<1)temp=0;
      mapa[ind]=temp;
      if ((i>fa-1) && (i<fa+nfils) && (j>ci-1) && (j<ci+ncols)){
/*10. Se fuerza a que para todos los índices que se encuentren dentro del área de
cobertura (los que se desean calcular) el valor de 'campo[ind]' sea igual a -9999*/
         campo[ind]=-9999;
      }
```

```
   }

   fclose(grid);
}

else{
   printf("No se puede abrir fichero de entrada\n");
   exit(1);
}

/***SE LEE LA CABECERA DEL FICHERO ALTURAS***/

if ((grid2=fopen(argv[6],"r"))!=NULL){

   tira2=(char*)calloc(80,sizeof(char));

   fscanf(grid2,"%s\n",tira2);
   fscanf(grid2,"%d\n",&ncols3);

   fscanf(grid2,"%s\n",tira2);
   fscanf(grid2,"%d\n",&nrows3);

   fscanf(grid2,"%s\n",tira2);
   lee_double(grid2,&xllcorner3);

   fscanf(grid2,"%s\n",tira2);
   lee_double(grid2,&yllcorner3);

   fscanf(grid2,"%s\n",tira2);
   lee_double(grid2,&cellsize3);

   fscanf(grid2,"%s\n",tira2);
   fscanf(grid2,"%d\n",&nodata3);

   free(tira2);


   /***RESERVA DE MEMORIA PARA EL MDT (MAPA) Y EL MAPA DE CAMPO (CAMPO) RESULTADO***/

   prod=(long) nrows1*ncols1;
   mapa2=(float*)calloc(prod,sizeof(float));

   if (!mapa2){
      printf("No queda memoria disponible para ciudad o campo\n");
      exit(1);
   }

   for (i=0;i<nrows1;i++)
   for (j=0;j<ncols1;j++){
```

```
      ind=(long) ncols1*i+j;
      lee_float(grid2,&temp);
      if(fabs(temp+9999)<1)temp=0;
      mapa2[ind]=temp;
   }

   fclose(grid2);
}

else{
   printf("No se puede abrir fichero de entrada\n");
   exit(1);
}


/***BUCLE PRINCIPAL***/

for (fr=0;fr<nrows1;fr++)
for (cr=0;cr<ncols1;cr++){ /***INICIO DEL BUCLE PRINCIPAL***/
   ind=(long)ncols1*fr+cr;

   if ((fr>fa-1) && (fr<fa+nfils) && (cr>ci-1) && (cr<ci+ncols) &&
   ((fr==fa) || (fr==fa+nfils-1) || (cr==ci) || (cr==ci+ncols-1) ) ){

/***INICIO DE LA CONDICIÓN DE CELDA EN LA QUE SE DESEA HALLAR LAS PÉRDIDAS***/
/*11. Se definen las nuevas condiciones para que sólo se levante el perfil (es decir, se
obtengan los vectores 'perfil', 'vector_distancias' e 'ind_calc') en las celdas de los
bordes del área de cobertura a calcular*/


      /***CÁLCULO DE LA DISTANCIA TX-RX***/

      dx=cr-ct;
      dy=fr-ft;
      e=(float)sqrt(dx*dx+dy*dy);
      R=e*(celda);
/*12. Se elimina la condición de devolver -9999 en caso de que figura=1 y R>radio*/


      /***RESERVA DE MEMORIA***/

      e1=floor(e)+1;
      x=(int*)calloc(e1,sizeof(int));
      y=(int*)calloc(e1,sizeof(int));
      if(!x||!y){
         printf("No queda memoria disponible para coordenadas\n");
         exit(1);
      }
```

```
      /***CREACIÓN DEL VECTOR X***/

      paso=(e==0) ? 0 : dx/floor(e);
      for (i=0;i<(e1-1);i++){
         aux=ct+i*paso-floor(ct+i*paso);
         if (aux<0.5) x[i]=floor(ct+i*paso);
         else x[i]=ceil(ct+i*paso);
      }

      x[e1-1]=cr;


      /***CREACIÓN DEL VECTOR Y***/

      paso=(e==0) ? 0 : dy/floor(e);
      for (i=0;i<(e1-1);i++){
         aux=ft+i*paso-floor(ft+i*paso);
         if (aux<0.5) y[i]=floor(ft+i*paso);
         else y[i]=ceil(ft+i*paso);
      }

      y[e1-1]=fr;


      /***SE HALLA EL PERFIL***/

      perfil=(float*)calloc(e1,sizeof(float));
      perfil2=(float*)calloc(e1,sizeof(float));
      ind_calc=(int*)calloc(e1,sizeof(int));
/*13. En 'ind_calc' se guardan los índices que componen el perfil completo de la
iteración actual (desde el transistor a cada una de las celdas que se encuentre en el
exterior del área de cobertura)*/

      if (!perfil){
         printf("No queda memoria disponible para perfil\n");
         exit(1);
      }

      if (!perfil2){
         printf("No queda memoria disponible para perfil\n");
         exit(1);
      }

      for(j=0;j<e1;j++){
         ind=(long) ncols1*y[j]+x[j];
         perfil2[j]=mapa2[ind];
         perfil[j]=mapa[ind];
         ind_calc[j]=ind;
/*14. Se rellena el array 'ind_calc' como se ha comentado anteriormente*/
```

```
      }
/*15. No se libera aquí el espacio de memoria de 'x' e 'y' como se hace en el primario,
puesto que ambas variables se pasarán como argumentos al método CalculaAtenuacion*/

      if (e1==1) {
         L=0;
         campo[ind]=L;
      }
/*16. Se añade este if para que en caso de que 'e1' sea igual a 1 (es decir, la celda de
la actual iteración es la del transmisor, que en este caso se encuentra en el borde del
área calculada), el valor de las pérdidas sea igual a 0 */

      else{
/***17. BUCLE PARA REALIZAR LA LLAMADA AL MÉTODO CALCULAATENUACION PARA AQUELLOS INDICES
QUE DEBO CALCULAR DENTRO DEL PERFIL COMPLETO DE ESTA ITERACIÓN***/
         ep=e1-1;
         for(i=(e1-1);i>0;i--){

            if(campo[ind_calc[ep]]==(-9999)){
               L=CalculaAtenuacion(perfil, perfil2, hrini, htini, e1, celda,
               eleccion, frecuencia, ep, R, x, y, ft, ct, mapa2, ind_calc,
               figura, radio );

               campo[ind_calc[ep]]=L - G;
            }

            ep=ep-1;
         }

      }

      free(x);
      free(y);
      free (perfil);
      free (perfil2);
/*18. Se libera el espacio de memoria de los vectores 'x', 'y', 'perfil' Y 'perfil2'
para cada iteración */
   } /***FIN DE LA CONDICIÓN DE CELDA EN LA QUE HEMOS CALCULADO LAS PÉRDIDAS***/

   else if ((fr<fa) || (fr>fa+nfils-1) || (cr<ci) || (cr>ci+ncols-1)){
/*19. Con esta condición, se asigna valor 'campo[ind]' a todas las celdas que estén
fuera del área de cobertura*/
      campo[ind]=-9999;
   }

} /***FIN DEL BUCLE PRINCIPAL***/


/***ESCRITURA DEL FICHERO DE CAMPO EN DISCO***/
```

```
if ((resultados=fopen(argv[3],"a"))!=NULL){

   filasar=((yllcorner2+cellsize2*nrows2)-(yllcorner1+cellsize1*nrows1))/cellsize2;
   filasab=nrows2-nrows1-filasar;
   coli=(xllcorner1-xllcorner2)/cellsize2;
   cold=ncols2-ncols1-coli;
   for (x1=0;x1<filasar;x1++){
      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }
      fprintf(resultados,"\n");
   }

   for (i=0;i<nrows1;i++){
      for (y1=0;y1<coli;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      for (j=0;j<ncols1;j++){
         ind1=(long) ncols1*i+j;
         escribe_float(resultados,&(campo[ind1]));
         fprintf(resultados," ");
      }

      for (y1=0;y1<cold;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   for (x1=0;x1<filasab;x1++){
      for (y1=0;y1<ncols2;y1++){
         fprintf(resultados,"-9999");
         fprintf(resultados," ");
      }

      fprintf(resultados,"\n");
   }

   fclose (resultados);
   printf("El campo está  en disco\n");
}

else{
   printf("No se puede abrir fichero de salida\n");
```

```
   exit (1);
}

free(campo);
free(mapa);
free (mapa2);
comprueba=fopen(argv[24],"w");
fprintf(comprueba,"\n");

fclose (comprueba);
end = clock();

if ((tiempo=fopen("c:\\arcgis\\RAGIS\\tiempo.txt","w"))!=NULL){
   fprintf(tiempo,"The time was: %f\n", (end - start) / CLK_TCK);
   fclose(tiempo);
}

} /***FIN DE MAIN***/
```

4.3.2. Fichero calculaatenuacion.cpp.

```
#include <dos.h>
#include <windows.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

#define PI abs(acos(-1.0))
#define Re 8.5e6

#define PI fabs(acos(-1.))
#define PI abs(acos(-1.0))


/***1.- MÉTODO OBP_WB***/

float obp_wb (float v) {

   float q;

   q=2.34*pow(v,0.9);
   return q;
}
```

```
/***2.- MÉTODO DIF_WB***/

float dif_wb (float lambda,float r,float beta) {

   float F;
   F=sqrt(lambda)/(2*PI*sqrt(r));
   F=F*fabs((1/beta)-(1/(2*PI+beta)));
   return F;
}


/***3.- MÉTODO CALCULAATENUACION***/

float CalculaAtenuacion(float *perfil, float *perfil2, float hrini, float htini, int e1,
float celda, int eleccion, float frecuencia, int ep, float R, int *x, int *y, int ft,
int ct, float *mapa2, int *ind_calc, float figura, float radio ) {
/*20. Se pasan como argumentos un gran número de variables de mapa.cpp*/

   float ht,hr;
   int *vec1;
   int posiciontx,fin,j_ini,n_nocero;
   float hmax,hmin,hmed,H,hu;
   int i,j;
   float d;
   int n_edif;
   int this_i,next_i,ultimo_edif;
   float raux;
   float lambda,alfaultimo,alfapenult,t;
   float r;
   float gamma_f;
   float hvar;
   float q,F;
   float ka,kd;

   float theta, r_xia,, lam, phi;
   float deltaHb, deltaHm, x_xia;
   float Lfs, Lrts, Lmd;

   float Lo,L_edif,L_dif,L;
/*21. Se declaran todas estas variables en este método, anteriormente se encontraban
declaradas en mapa.cpp*/
   int ep1;
/*22. Se declara 'ep1'*/
   float *perfil_actual;
   float *perfil_actual2;
/*23. Se declaran los nuevos vectores 'perfil_actual' y 'perfil_actual2' a reutilizar en
cada llamada a método*/
```

```
   lambda=300/frecuencia;
/*24. Se obtiene 'lambda'*/
   ep1=ep+1;
/*25. 'ep1' es lo mismo que 'e1', pero para cada una de las iteraciones de 'ep'*/

   perfil_actual = (float*)calloc(ep1,sizeof(float));
   perfil_actual2 = (float*)calloc(ep1,sizeof(float));
/*26. Se reserva memoria para los nuevos vectores*/

   if ((ep1==e1) && (mapa2[ind_calc[ep]]==0) ){
/*27. Creación de los vectores 'perfil_actual' y 'perfil_actual2' para las celdas
exteriores del área de cobertura a calcular, y cumpliéndose que la celda (que pertenece
al receptor) no pertenece a un edificio (es decir, se encuentra en la calle)*/
      for (i=0; i<ep1; i++){
         perfil_actual[i]=perfil[i];
         perfil_actual2[i]=perfil2[i];
      }

   }

   else if (mapa2[ind_calc[ep]]==0){
/*28. Creación de los vectores 'perfil_actual' y 'perfil_actual2' para las celdas
interiores del área de cobertura a calcular*/
      R=sqrt((pow((y[ep]-ft)*celda,2))+(pow((x[ep]-ct)*celda,2)));
      for (i=0; i<ep1; i++){
         perfil_actual[i]=perfil[i];
         perfil_actual2[i]=perfil2[i];
      }

   }

   else{
/*29. Se devuelve -9999 en caso de que el receptor no se encuentre en la calle*/
      return -9999;
   }

   if ((figura==1) && (R > radio)){
/*30. Bucle if con el que se devuelve -9999 y se libera la memoria reservada con calloc
al principio de la llamada, en caso que se quiera calcular un área de cobertura
circular, y dicha altura del MDT no pertenezca a esa circunferencia de cálculo*/
      free(perfil_actual);
      free(perfil_actual2);

      return -9999;
   }

   hr = hrini + perfil_actual[ep1-1];
   ht = htini + perfil_actual[0] - perfil_actual2[0];
```

```
/***SE DETERMINA SI EL TX ESTÁ EN LA CALLE O EN UN EDIFICIO***/
if (perfil_actual2[0]==0) posiciontx=0;
else posiciontx=1;


/***ALMACENAMIENTO DE LOS ÍNDICES DEL PERFIL EN LOS QUE HAY EDIFICIOS***/

vec1=(int*) calloc(e1,sizeof(int));
if (!vec1){
   printf("No queda memoria disponible para vec1\n");
   exit(1);
}

j_ini=fin=0;
if (posiciontx==1)
do{
   if (perfil_actual2[j_ini++]==0) fin=1;
}

while (fin==0);
hmed=hu=0;
for (j=j_ini,n_nocero=0;j<ep1;j++){
   if (perfil_actual2[j]>0){
      if (n_nocero==0) hmax=hmin=perfil_actual[j];
      vec1[n_nocero++]=j;
      hmed+=perfil_actual[j];
      hu=perfil_actual[j];
      if (hu<hmin) hmin=hu;
      if (hmax<hu) hmax=hu;
   }

}

if (n_nocero==0) goto vision_directa;
else hmed=hmed/n_nocero;
H=ht-hmed;


/*CÁLCULO DE LA DISTANCIA MEDIA ENTRE EDIFICIOS*/

d=0;
n_edif=ultimo_edif=0;
for (j=0;j<(n_nocero-2);j++){
   this_i=vec1[j];
   next_i=vec1[j+1];
   if (perfil_actual2[this_i]>perfil_actual2[next_i]){
      n_edif+=1;
      if (ultimo_edif!=0) d+=(this_i-ultimo_edif)*celda;
```

```
         ultimo_edif=this_i;
      }

      else if ((next_i-this_i)>1){
         n_edif+=1;
         if (ultimo_edif!=0) d+=(this_i-ultimo_edif)*celda;
         ultimo_edif=this_i;
      }

   }
         if (ultimo_edif!=0) d+=(vec1[n_nocero-1]-ultimo_edif)*celda;
         d=(n_edif>0)? d/n_edif : 0;


   /***DISTANCIA 2D ÚLTIMO EDIFICIO-RX (M)***/

   raux=((ep1-1)-vec1[n_nocero-1])*celda;
   /***ÁNGULOS DE INCIDENCIA DEL RAYO SOBRE EL ÚLTIMO Y PENÚLTIMO EDIFICIO (RAD)
   TENIENDO EN CUENTA LA CURVATURA DE LA TIERRA***/

   alfaultimo=atan(((ht-hu)/R)-(R/(2*Re)));
   alfapenult=atan((H/R)-(R/(2*Re)));


   /***PARÁMETRO T***/

   t=alfapenult*sqrt(d/lambda)*sqrt(PI);


   /***DISTANCIA 3D ÚLTIMO EDIFICIO-RX (M)***/

   r=sqrt(pow(raux,2)+pow(hu-hr,2));


   /***ÁNGULO DE DIFRACCIÓN FINAL (RAD)***/

   gamma_f=(raux!=0) ? atan((hmed-ht)/raux) : atan ((hmed-hr)/celda);

   free(vec1);
   free(perfil_actual);
   free(perfil_actual2);


         /***CÁLCULO DE LAS PÉRDIDAS (DB)***/


/////***ESPACIO LIBRE***/////

   if (R>0) Lo=-20*log10(lambda/(4*PI*R));
```

```
   else Lo=0.0;
//////////////////////////


/////***MODELO DE WALFISCH-BERTONI***/////

   if (eleccion==1){
/*31. El argumento introducido en ArcMap para elegir el método urbano de cálculo de
pérdidas, se obtiene a partir de la variable 'eleccion', cuyo valor lo ha obtenido en
mapa.cpp*/
      if (n_edif>0){
         q=obp_wb(t/sqrt(PI));
         if (q>0) L_edif=-20*log10(q)+0.525*hvar;
         else L_edif=0;
      }

      else L_edif=0;
      if (hmed < hr)
      F = 1;
      else{
         F=dif_wb(lambda,r,gamma_f-alfaultimo);
         F=F*sqrt(2);
      }

      L_dif=-20*log10(F);
   }
//////////////////////////////////////


/////***MODELO COST-231***/////

   if (eleccion==2){
/*32. De nuevo, la variable 'eleccion' señala a este método como elegido para el
cálculo*/
      if (n_edif>0) {
         if (H<0) {
            if (R/1000<0.5) ka = 54 - 0.8*H*(R/1000)/0.5;
            else ka = 54 - 0.8*H;
         }

         else ka = 54;
         if ((H<0)&&(hmed!=hr)) kd = 18 - 15*H/(hmed - hr);
         else kd = 18;
         if (fabs(1 + H)==0)
         L_edif = -18*log10 (0.0000000001) + ka + kd*log10(R/1000) –
         (4 + 0.7*(frecuencia/925 - 1))*log10(frecuencia) - 9*log10(d);

         else
         L_edif = -18*log10 (fabs(1 + H)) + ka + kd*log10(R/1000) –
```

```
         (4 + 0.7*(frecuencia/925 - 1))*log10(frecuencia) - 9*log10(d);
         if (hmed <= hr)
         L_dif = 0;
         else
         L_dif =  -16.9 -10*log10(raux+0.00001) + 10*log10 (frecuencia)+
         20*log10(fabs(hmed - hr));
      }

      else {
         L_edif = 0;
         if (hmed <= hr)
         L_dif = 0;
         else
         L_dif =  -16.9 -10*log10(raux+0.00001) + 10*log10(frecuencia)
         + 20*log10(fabs(hmed - hr));

      }

   }
//////////////////////////////////////////


/////***MODELO XIA-BERTONI***/////

   if (eleccion==3){
/*33. Tercer posible valor de la variable 'eleccion */
      x_xia=raux+0.00001;
      lam = 3e8/(frecuencia*1e6);
      deltaHb=ht-hmed;
      deltaHm=hmed-hr;

      if (n_edif>0) {
         theta = atan(deltaHm/x_xia);
         if (theta<=0) theta = 0.02;
         r_xia = sqrt(pow(deltaHm,2)+pow(x_xia,2));
         phi=-atan(deltaHb/d);


   /***TRES SUPUESTOS PARA EL MODELO XIA-BERTONI***/

   ///CASO 1: ANTENAS TRANSMISORAS CERCA DEL NIVEL EDIO DE LOS EDIFICIOS

            if((ht>=(hmed-0.5))&&(ht<=(hmed+0.5))){
              Lfs = -10*log10(pow((lam/(2*sqrt(2)*PI*R)),2));
               if (hmed < hr)
               Lrts = 0;
               else
               Lrts=-10*log10((lam/(2*pow(PI,2)*r_xia))*pow(((1/theta)-1/(2*PI+theta))
               ,2));
```

```
            Lmd = -10*log10(pow(d/R,2));
         }

   ///CASO 2: ANTENAS TRANSMISORAS POR ENCIMA DEL NIVEL EDIO DE LOS EDIFICIOS

         else if(ht>hmed){
            Lfs = -10*log10(pow((lam/(4*PI*R)),2));
            if (hmed < hr)
            Lrts = 0;
            else
            Lrts=-10*log10((lam/(2*pow(PI,2)*r_xia))*pow(((1/theta)-1/(2*PI+theta))
            ,2));
            Lmd = -10*log10(pow(2.35,2)*pow(((deltaHb/R)*sqrt(d/lam)),1.8));
         }

   ///CASO 3: ANTENAS TRANSMISORAS POR DEBAJO DEL NIVEL EDIO DE LOS EDIFICIOS

         else if(ht<hmed){
            Lfs = -10*log10(pow((lam/(2*sqrt(2)*PI*R)),2));
            if (hmed < hr)
            Lrts = 0;
            else
            Lrts=-10*log10((lam/(2*pow(PI,2)*r_xia))*pow(((1/theta)-1/(2*PI+theta))
            ,2));
            Lmd=-10*log10(pow((d/(2*PI*(R-d))),2)*(lam/sqrt(pow(deltaHb,2)+pow(d,2)))
            * pow((1/phi-1/(2*PI+phi)),2));
         }

      }

      else {

      Lmd=Lrts=0;

      if (x_xia>0) {
         theta = atan(deltaHm/x_xia);
         if (theta<=0) theta = 0.02;
         r_xia = sqrt(pow(deltaHm,2)+pow(x_xia,2))+0.00001;
         if (hmed < hr)
         Lrts = 0;
         else
         Lrts=-10*log10((lam/(2*pow(PI,2)*r_xia))*pow(((1/theta)- 1/(2*PI+theta))
         ,2));
      }

   ///CASO 1: ANTENAS TRANSMISORAS CERCA DEL NIVEL EDIO DE LOS EDIFICIOS

.........if((ht>=(hmed-0.5))&&(ht<=(hmed+0.5))){
```

```
            Lfs = -10*log10(pow((lam/(2*sqrt(2)*PI*R)),2));
         }

   ///CASO 2: ANTENAS TRANSMISORAS POR ENCIMA DEL NIVEL EDIO DE LOS EDIFICIOS

         else if (ht>hmed) {
            Lfs = -10*log10(pow((lam/(4*PI*R)),2));
         }

   ///CASO 3: ANTENAS TRANSMISORAS POR DEBAJO DEL NIVEL EDIO DE LOS EDIFICIOS

         else if (ht<hmed) {
            Lfs = -10*log10(pow((lam/(2*sqrt(2)*PI*R)),2));
         }

      }

      L = Lfs + Lrts + Lmd;

      goto fin_bucle;
   }
////////////////////////////////////////
   L=Lo+L_edif+L_dif;
   goto fin_bucle;


   /***VISIÓN DIRECTA***/

   vision_directa:

   if (R>0){
      R=sqrt(R*R+(ht-hr)*(ht-hr));
      L=-20*log10(lambda/(4*PI*R));
   }

   else L=0;
   fin_bucle:

   return L;
```

4.4. MODIFICACIÓN Y PARTICULARIDADES DEL MODELO URBANO.

Como ya se ha mencionado en más de una ocasión, el objetivo de la optimización se centra en el trabajo realizado sobre el levantamiento del perfil en cada uno de los proyectos (rural y urbano). Ya se sabe que tanto el funcionamiento como las modificaciones al respecto son similares. Como en el apartado 1.4. se ha explicado todo con el mayor grado de detalle posible, para no caer en información redundante, en el actual punto del proyecto se centrará la atención en comentar aquellas modificaciones y funcionamientos que varían respecto del modelo rural.

En el proyecto primario, todo el código se encuentra almacenado en el mismo archivo "mapa.cpp". La primera acción realizada es **crear una estructura de proyecto similar a la del modelo rural**; para ello se ha creado un nuevo archivo llamado "calculaatenuacion.cpp" que estará formado por un método con su mismo nombre, *CalculaAtenuacion*.

El método *CalculaAtenuacion* es similar al del proyecto rural; a él se llamará desde *main* para calcular las pérdidas a lo largo de cada una de las celdas almacenadas dentro del vector *perfil* (creado como ya se sabe en el método *main*). Dentro de este nuevo método se desarrolla tanto el método de cálculo de perdidas COST-231, como el de Xia/Bertoni y Walfisch/Bertoni.

Una vez conseguida la estructura de código deseada, se procede a realizar las modificaciones necesarias.

4.4.1. Modificación en el fichero mapa.cpp.

En la primera modificación, por lo que se acaba de comentar, se define el método *CalculaAtenuacion*, eliminando por otra parte las declaraciones de los métodos

opb_wb y *dif_wb*, ya que estos serán llamados desde dicho método *CalculaAtenuacion*, y no desde *main*.

```
main (int argc,char *argv[]){

float CalculaAtenuacion(float *perfil, float *perfil2, float hrini, float htini, int e1,
float celda, int eleccion, float frecuencia, int ep, float R, int *x, int *y, int ft,
int ct, float *mapa2, int *ind_calc, float figura, float radio );
/*1. Se declara el método CalculaAtenuacion */
```

En cuanto a la declaración de variables posterior a la modificación 1, se eliman todas aquellas que se crearán dentro de *CalculaAtenuacion* a partir de la llamada a este método. Son un gran número de ellas: *vec1*, *posiciontx*, *fin*, *j_ini*, *n_nocero*, *hmax*, *hmin*, *hmed*, *H*, *hu*, *this_i*, *next_i*, *ultimo_edif*, *d*, *raux*, *q*, *F*, *ka*, *kd*, *ht*, *hr*, *hvar*, *n_edif*, *lambda*, *alfaultimo*, *alfapenult*, *t*, *r*, *gamma_f*, *l_edif*, *l_dif*, *Lo*.

Además, al igual que en el rural, se añaden en el método *main* las variables necesarias para lograr el funcionamiento deseado: *ind_calc* y *ep*.

Dos diferencias a destacar del modelo urbano respecto al rural; la primera es la inexistencia del *vector_distancias* en el método *main*; y la segunda es que no se pasan como argumentos desde *RADIOGIS* las alturas físicas (mástiles, personas, coches, etc.) de las antenas transmisora (*ht*) y receptora (*hr*), sino que serán esas mismas, pero sumándoles también la altura del grid urbano (*htini* y *hrini* respectivamente). Esto influirá en el transmisor, el cual se encontrará sobre un edificio, pero no sobre el receptor, ya que éste circulará por la calle (altura de valor 0 en el grid urbano).

```
htini=atof(argv[8]);
hrini=atof(argv[9]);
```

Por otro lado, se ha incluido una parte de código para que cuando se introduzca como argumento desde *ArcMap* el método de predicción de pérdidas a utilizar, éste se almacene en una nueva variable llamada *eleccion*, la cual será pasada como argumento a su vez al método *CalculaAtenuacion*, que es donde realmente será utilizada.

```
if (!strcmp(argv[4],"wb")){
   eleccion=1;
}
/*5. La variable 'eleccion' será igual a 1 si en el entorno gráfico ArcMap se elige que
el cálculo de pérdidas sea realizado con el método Walfisch-Bertoni*/

else if (! strcmp(argv[4],"cost")){
   eleccion=2;
}
/*6. La variable 'eleccion' será igual a 2 si en el entorno gráfico ArcMap se elige que
el cálculo de pérdidas sea realizado con el método COST-231*/

if (!strcmp(argv[4],"xia")){
   eleccion=3;
}
/*7. La variable 'eleccion' será igual a 3 si en el entorno gráfico ArcMap se elige que
el cálculo de pérdidas sea realizado con el método Xia-Bertoni*/
```

Para finalizar, destacar que en el proyecto urbano se puede trabajar con uno o dos grids; es decir, se pueden calcular las pérdidas sobre un MDT formado únicamente por un grid urbano (equivaldría a una ciudad al nivel del mar), o por éste mismo más la suma de las alturas de un grid rural. Es conveniente realizar una aclaración sobre a qué se refieren una serie de variables en el código relacionadas con este asunto:

```
/***FICHEROS UTILIZADOS***/

FILE *grid; //rural+urbano
FILE *grid2; //urbano
FILE *fdatos;
FILE *resultados;
FILE *mdt; //rural + urbano
FILE *comprueba;
FILE *tiempo;
```

- *ncols2*, *nrows2*, *nodata2*, *xllcorner2*, *yllcorner2*, *cellsize2* → parámetros del fichero que contiene el MDT sobre el que se va a trabajar. Estas variables sólo sirven para copiar los datos en el fichero resultados.

- *mapa*, *grid*, *ncols1*, *nrows1*, *nodata1*, *xllcorner1*, *yllcorner1*, *cellsize1* → variables utilizadas cuando el grid está formado por la suma de los terrenos rural y urbano.

- *mapa2*, *grid2*, *ncols3*, *nrows3*, *nodata3*, *xllcorner3*, *yllcorner3*, *cellsize3* → variables utilizadas cuando el grid está formado únicamente por el terreno urbano (0 metros equivale a una calle, mientras que las otras diferentes alturas estarán referidas a edificios).

Como detalle, decir que los valores de *ncols1*, *nrows1*, *nodata1*, *xllcorner1*, *yllcorner1*, *cellsize1* son iguales a los de *ncols3*, *nrows3*, *nodata3*, *xllcorner3*, *yllcorner3*, *cellsize3* respectivamente. Esto se da por el hecho de que cuando se trabaje con un grid rural adicional, éste será recortado por *RADIOGIS* y quedará del mismo tamaño que el del grid urbano, que será el predominante.

4.4.2. Modificación en el fichero calculaatenuacion.cpp.

Una vez creado el vector *perfil* en el método *main*, se procede a calcular las pérdidas en *CalculaAtenuacion*. Pero previo al cálculo habrá que adaptar una serie de variables dependiendo de la situación del receptor considerado en cada iteración. Hay que recordar que sólo se llamará al método *CalculaAtenuacion* en caso de que no se hayan calculado las pérdidas para la correspondiente celda previamente. Las adaptaciones son similares a las del proyecto rural, aunque contienen algunas salvedades.

En la primera de las condiciones se completan los vectores *perfil_actual* (alturas del grid rural) y *perfil_actual2* (alturas del grid urbano), los cuales serán idénticos que los vectores del perfil calculados en *main* antes de iniciar las llamadas al actual método. Esta afirmación es correcta porque en este caso, el receptor se encuentra sobre los extremos de la matriz en la que se desea calcular las pérdidas (*ep1 = e1*), y además, se cumplirá que la altura de *mapa2* para esta celda es igual a *0*, o lo que es lo mismo, el receptor se encuentra en la calle, y no sobre un edificio.

Nota: tener en cuenta que, como se comentó en el modelo rural, por estar en los extremos no se cumple siempre que se vayan a calcular las pérdidas en ellas, ya que si el

transmisor también se encuentra en el borde del área, o cercano a él, puede que esa celda ya haya formado parte de otro perfil, y no proceda calcular las pérdidas en este caso.

```
   if ((ep1==e1) && (mapa2[ind_calc[ep]]==0) ){
/*28. Creación de los vectores 'perfil_actual' y 'perfil_actual2' para las celdas
exteriores del área de cobertura a calcular, y cumpliéndose que la celda (que pertenece
al receptor) no pertenece a un edificio (es decir, se encuentra en la calle)*/
      for (i=0; i<ep1; i++){
         perfil_actual[i]=perfil[i];
         perfil_actual2[i]=perfil2[i];
      }

   }
```

Para el caso general en que las celdas del receptor no pertenecen al borde del área de cobertura, las acciones a realizar son más simples que para el proyecto rural, puesto que ahora *vector_distancias*, *vector_distancias_actual* y *Rt* no existen. Únicamente se recalcula la distancia total sobre el suelo y el actual receptor (*R*), y se completan los nuevos vectores *perfil_actual* y *perfil_actual2* hasta la posición que pertenezca a la actual iteración.

```
   else if (mapa2[ind_calc[ep]]==0){
/*29. Creación de los vectores 'perfil_actual' y 'perfil_actual2' para las celdas
interiores del área de cobertura a calcular*/
      R=sqrt((pow((y[ep]-ft)*celda,2))+(pow((x[ep]-ct)*celda,2)));
      for (i=0; i<ep1; i++){
         perfil_actual[i]=perfil[i];
         perfil_actual2[i]=perfil2[i];
      }

   }
```

Como última situación, puede darse que en la actual iteración el receptor caiga sobre un edificio, en cuyo caso no se calculan las pérdidas. Se puede observar como en el anterior bloque, la condición de entrada es que el receptor se encuentre en la calle (*mapa2[ind_calc[ep]]==0*).

```
   else{
```

```
/*30. Se devuelve -9999 en caso de que el receptor no se encuentre en la calle*/
      return -9999;
   }
```

Tras estas porciones de código, se realiza la condición ya comentada en el apartado 1.4.1. para el caso de que el área a calcular sea circular. Se devuelve valor nulo cuando el radio del radioenlace de la correspondiente iteración (*R*) sea mayor que el pasado como argumento (*radio*). La diferencia con el proyecto rural es que se libera de memoria un menor número de vectores, puesto que los únicos creados hasta el momento en el método *CalculaAtenuacion* han sido *perfil_actual* y *perfil_actual2*.

```
   if ((figura==1) && (R > radio)){
/*31. Bucle if con el que se devuelve -9999 y se libera la memoria reservada con calloc
al principio de la llamada, en caso que se quiera calcular un área de cobertura
circular, y dicha altura del MDT no pertenezca a esa circunferencia de cálculo*/
      free(perfil_actual);
      free(perfil_actual2);

      return -9999;
   }
```

Para finalizar, comentar algunos detalles como la creación de *lambda* en el método *CalculaAtenuacion* y no en *main*, ya que es en el primero de estos donde se utiliza ahora; o la diferencia que existe en las alturas consideradas del transmisor y receptor con respecto al proyecto rural.

```
   hr = hrini + perfil_actual[ep1-1];
   ht = htini + perfil_actual[0] - perfil_actual2[0];
```

A la altura *htini* (mástil de la antena más altura del edificio), pasada como argumento en *RADIOGIS*, se le suma la altura del grid rural (en el caso que se haya pasado como argumento), y se resta la del urbano; es decir, *ht* es la suma del mástil de la antena transmisora, más la altura del terreno con respecto al nivel del mar (no se tiene en cuenta la altura del edificio sobre el que se encuentre).

Por otra parte, *hrini* simplemente es la altura del receptor (su valor típico es de 2 metros), por lo que *hr* será la misma, pero sumándole la altura del grid rural. En este

caso no se resta la altura de ningún edificio, porque como se ha explicado anteriormente, a este punto del código sólo llegarán los receptores que se encuentren en la calle (*pefil_actual2 [ep1 - 1] = 0*).

4.5. RESULTADOS.

4.5.1. Introducción.

Los pasos para realizar la simulación son idénticos a los explicados en el apartado 1.5. del modelo rural.

Ahora, el grid que se debe de cargar en *ArcMap* es el de un entorno urbano, en este caso, por ejemplo, uno con resolución 6 metros ("urbano_6grid.aux").

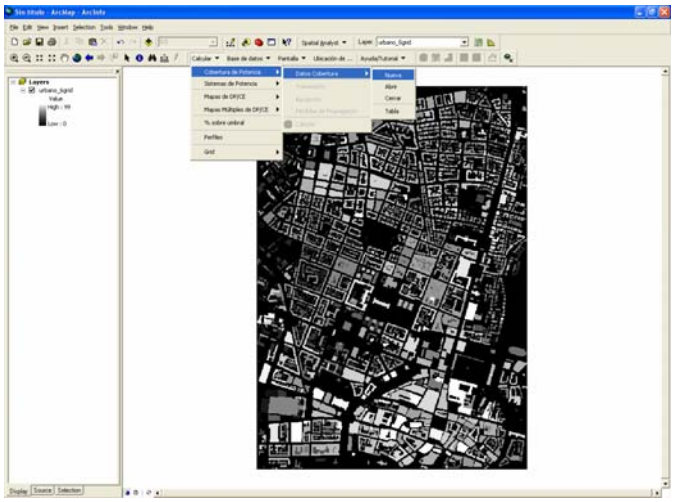
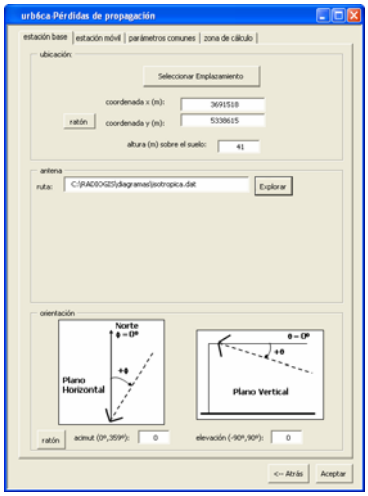


Figura 4.4. Creación de una cobertura urbana.

Los parámetros de recepción y transmisión, en cuanto al PIRE, ganancia, etc., pueden ser idénticos a los del rural.



En la "estación base", al definir la altura del transmisor sobre el suelo, se incluye el mástil y la antena (igual que en el caso rural), y además la altura del edificio sobre la que dicho conjunto está instalado.

Figura 4.5. Creación de una cobertura urbana. Pérdidas de propagación. Estación base.

La pestaña de la "estación móvil" se completa de igual manera que en el rural, dando la altura que el receptor tiene sobre el suelo,y como media se puede establecer 2 metros.

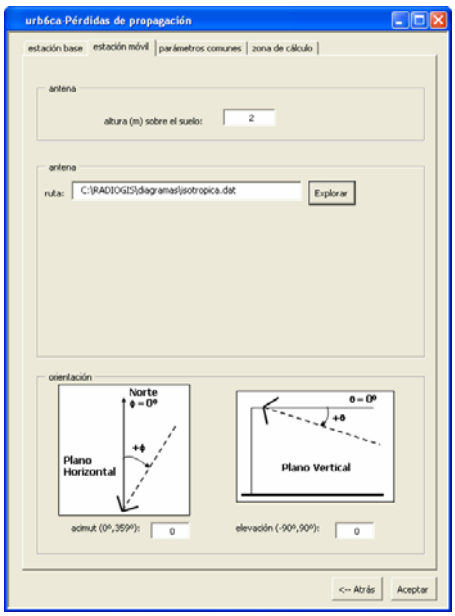


Figura 4.6. Creación de una cobertura urbana. Pérdidas de propagación. Estación móvil.

Nota: el modelo implica que la predicción de pérdidas se realizará sobre un receptor en el suelo, por lo que se descarta que éste pudiese estar en un edificio.

La diferencia más importante entre ambos modelos se encuentra en los "parámetros comunes".

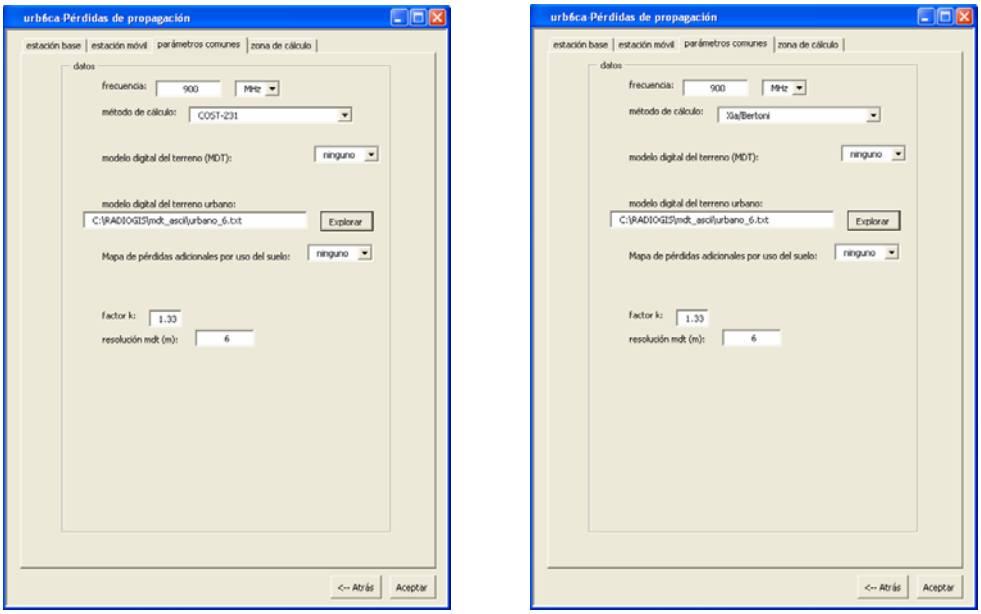


Figura 4.7. Creación de una cobertura urbana. Pérdidas de propagación. Parámetros comunes.

Se fija la frecuencia del radioenlace, así como el método de cálculo ("COST-231", "Xia-Bertoni" o "Walfisch-Bertoni"), el factor "k" y la resolución del MDT.

Sin embargo, ahora se pueden cargar dos modelos digitales del terreno. Un primero (modelo digital del terreno rural) se corresponde al mismo mapa cargado en el rural; y un segundo (modelo digital del terreno urbano) representa la altura de los diferentes edificios que se encuentran en la ciudad considerada.

En las simulaciones realizadas, sólo se tiene en cuenta el MDT rural, que se cargará en el programa en forma de grid, y de archivo de texto.

Por último, se establece el área de cálculo de la cobertura circular o rectangular.

Figura 4.8. Creación de una cobertura urbana. Pérdidas de propagación. Zonas de cálculo rectangular y circular.

Los resultados que se muestran a continuación, se han realizado para simulaciones del método COST-231. Para los otros modelos se obtienen datos similares de coberturas y tiempos de cálculo.

4.5.2. MDT resolución 6.

4.5.2.1. Área rectangular.

MDT6GRID											
LADO (m)	ÁREA (m^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
96	9216	0,015	0,015	0,000	0,000	0	0,0	0	0,0	0	0,0
192	36864	0,016	0,016	0,000	0,000	0	0,0	0	0,0	0	0,0
288	82944	0,015	0,047	0,032	68,085	0	0,0	0	0,0	0	0,0
384	147456	0,031	0,079	0,048	60,759	0	0,0	0	0,1	0	0,0
480	230400	0,047	0,140	0,093	66,429	0	0,0	0	0,1	0	0,1
576	331776	0,078	0,203	0,125	61,576	0	0,1	0	0,2	0	0,1
672	451584	0,109	0,313	0,204	65,176	0	0,1	0	0,3	0	0,2
768	589824	0,141	0,438	0,297	67,808	0	0,1	0	0,4	0	0,3
864	746496	0,171	0,578	0,407	70,415	0	0,2	0	0,6	0	0,4
960	921600	0,219	0,766	0,547	71,410	0	0,2	0	0,8	0	0,5
1056	1115136	0,265	1,031	0,766	74,297	0	0,3	0	1,0	0	0,8
1152	1327104	0,312	1,297	0,985	75,944	0	0,3	0	1,3	0	1,0
1248	1557504	0,391	1,672	1,281	76,615	0	0,4	0	1,7	0	1,3
1344	1806336	0,453	2,125	1,672	78,682	0	0,5	0	2,1	0	1,7
1440	2073600	0,531	2,828	2,297	81,223	0	0,5	0	2,8	0	2,3
1536	2359296	0,625	3,375	2,750	81,481	0	0,6	0	3,4	0	2,8
1632	2663424	0,735	4,032	3,297	81,771	0	0,7	0	4,0	0	3,3
1728	2985984	0,828	4,812	3,984	82,793	0	0,8	0	4,8	0	4,0
1824	3326976	0,937	5,594	4,657	83,250	0	0,9	0	5,6	0	4,7
1920	3686400	1,078	6,500	5,422	83,415	0	1,1	0	6,5	0	5,4

Tabla 4.1. Tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 6 metros.

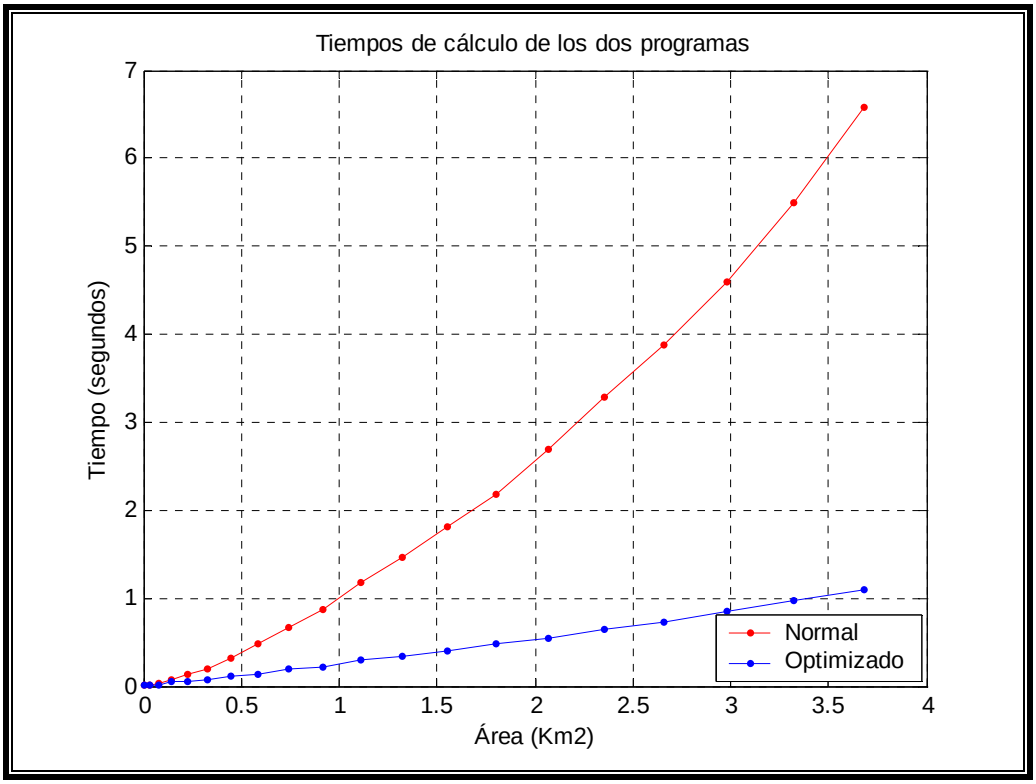


Figura 4.9. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 6 metros.

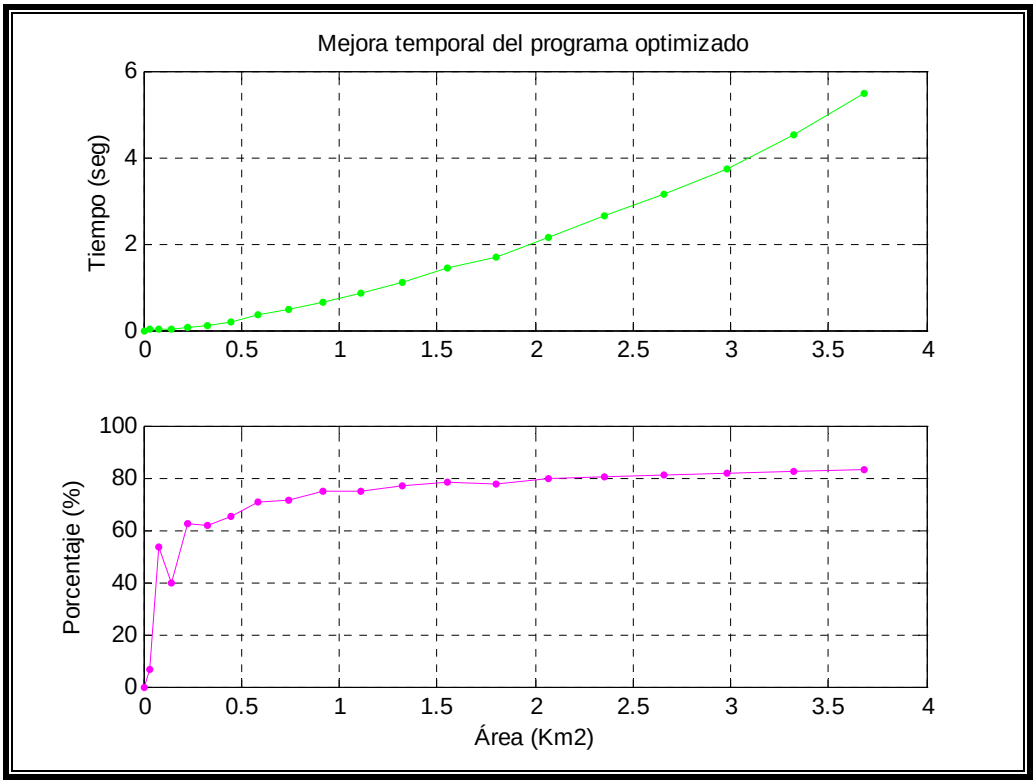


Figura 4.10. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 6 metros.

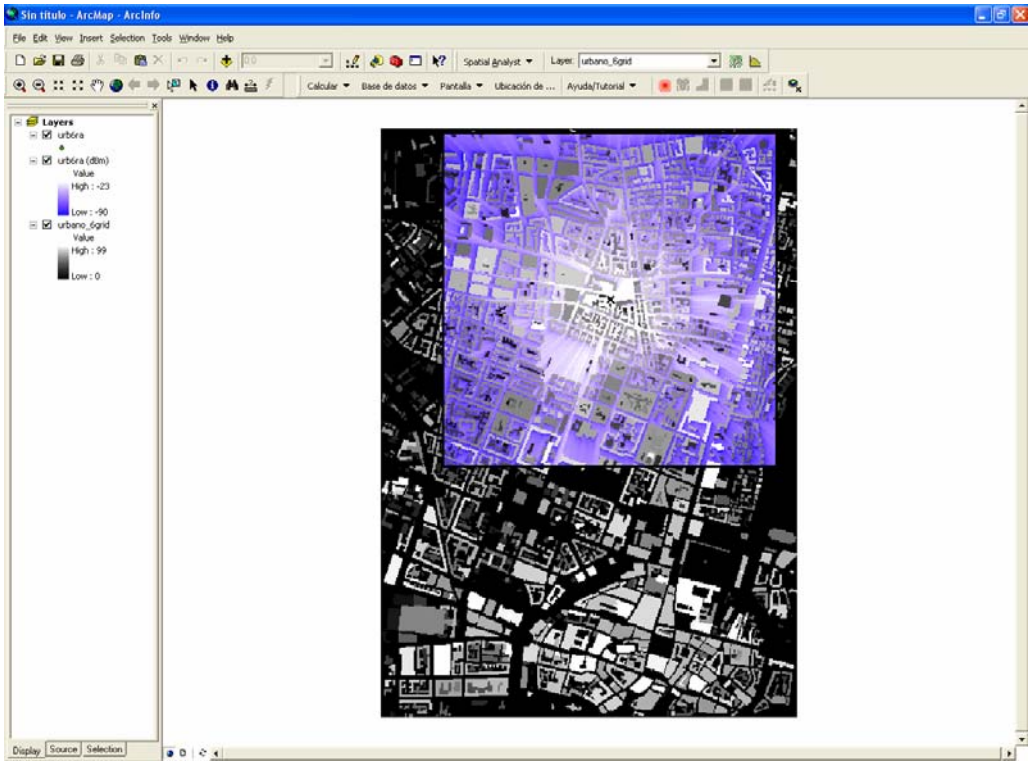


Figura 4.11. Pérdidas calculadas por el programa modificado sobre un área rectangular, para un MDT urbano de resolución 6 metros.

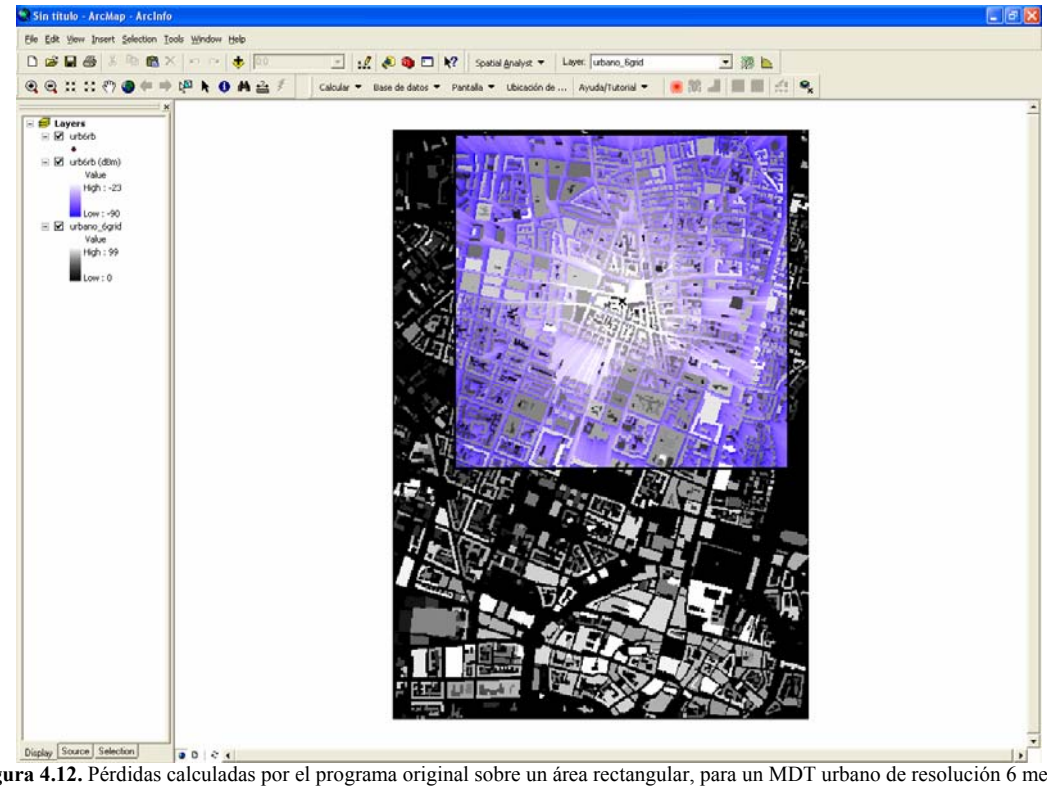


Figura 4.12. Pérdidas calculadas por el programa original sobre un área rectangular, para un MDT urbano de resolución 6 metros.

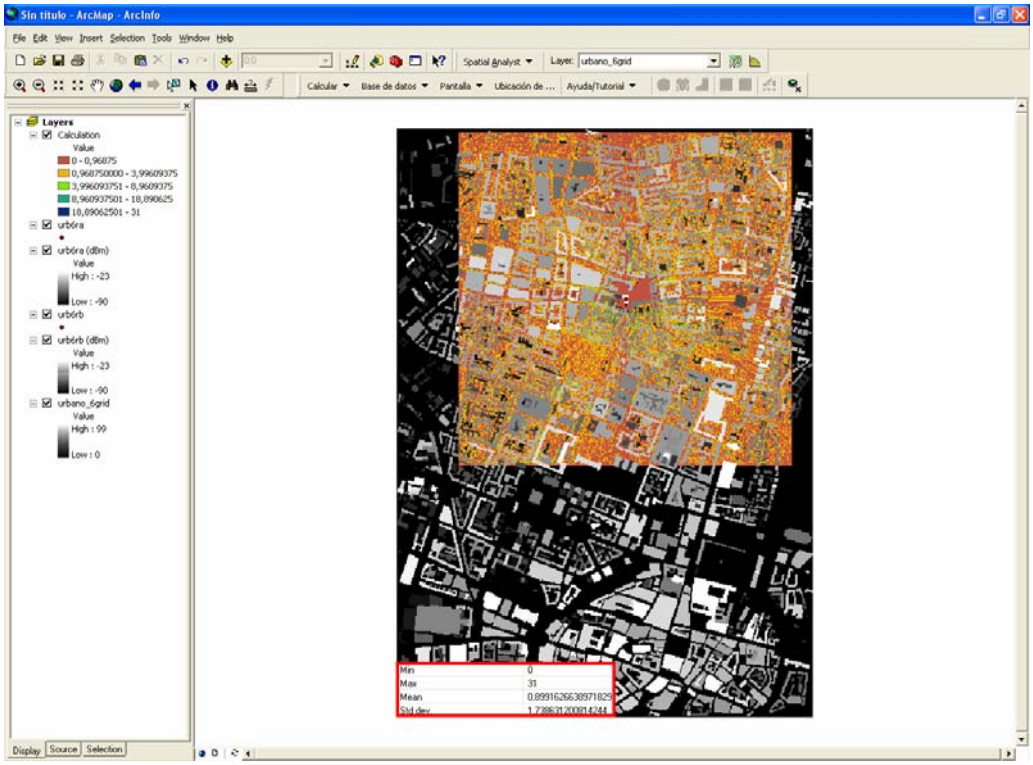


Figura 4.13. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT urbano de resolución 6 metros.

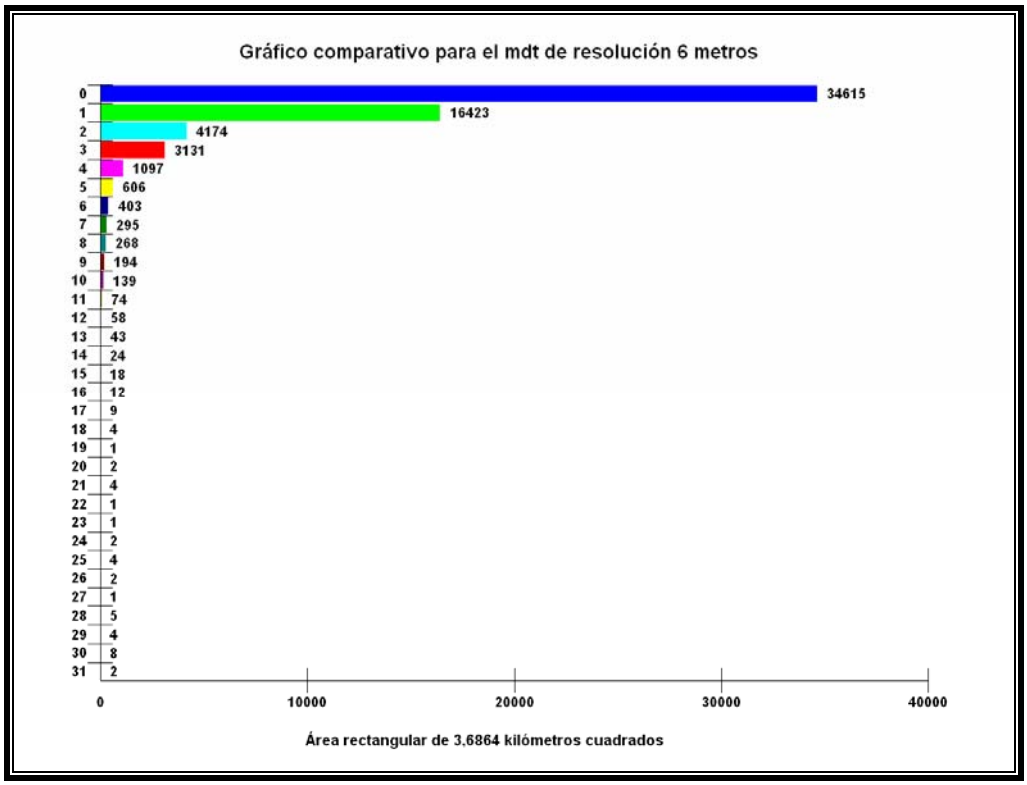


Figura 4.14. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT urbano de resolución 6 metros.

4.5.2.2. Área circular.

MDT6GRID											
RADIO (m)	ÁREA (m^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
48	2304	0,016	0,016	0,000	0,000	0	0,0	0	0,0	0	0,0
96	9216	0,016	0,016	0,000	0,000	0	0,0	0	0,0	0	0,0
144	20736	0,031	0,032	0,001	3,125	0	0,0	0	0,0	0	0,0
192	36864	0,032	0,063	0,031	49,206	0	0,0	0	0,1	0	0,0
240	57600	0,047	0,094	0,047	50,000	0	0,0	0	0,1	0	0,0
288	82944	0,078	0,157	0,079	50,318	0	0,1	0	0,2	0	0,1
336	112896	0,109	0,219	0,110	50,228	0	0,1	0	0,2	0	0,1
384	147456	0,125	0,312	0,187	59,936	0	0,1	0	0,3	0	0,2
432	186624	0,172	0,438	0,266	60,731	0	0,2	0	0,4	0	0,3
480	230400	0,203	0,563	0,360	63,943	0	0,2	0	0,6	0	0,4
528	278784	0,250	0,734	0,484	65,940	0	0,3	0	0,7	0	0,5
576	331776	0,296	0,938	0,642	68,443	0	0,3	0	0,9	0	0,6
624	389376	0,375	1,172	0,797	68,003	0	0,4	0	1,2	0	0,8
672	451584	0,453	1,469	1,016	69,163	0	0,5	0	1,5	0	1,0
720	518400	0,515	2,000	1,485	74,250	0	0,5	0	2,0	0	1,5
768	589824	0,594	2,328	1,734	74,485	0	0,6	0	2,3	0	1,7
816	665856	0,687	2,766	2,079	75,163	0	0,7	0	2,8	0	2,1
864	746496	0,797	3,281	2,484	75,709	0	0,8	0	3,3	0	2,5
912	831744	0,969	3,875	2,906	74,994	0	1,0	0	3,9	0	2,9
960	921600	1,031	4,484	3,453	77,007	0	1,0	0	4,5	0	3,5

Tabla 4.2. Tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 6 metros.

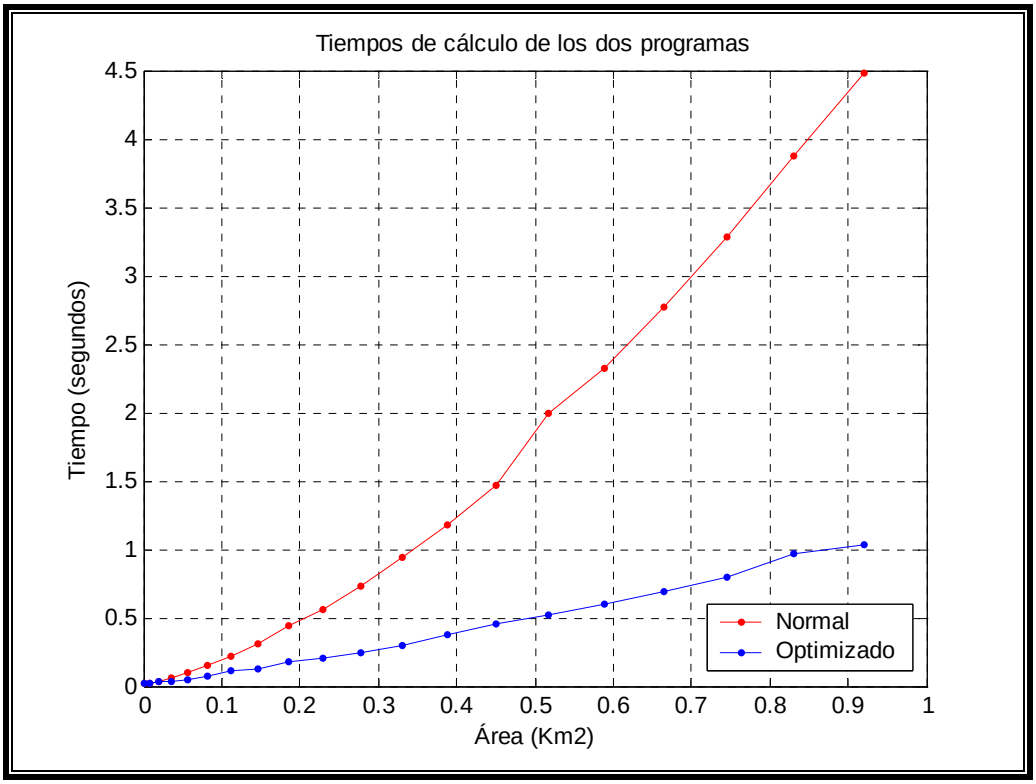


Figura 4.15. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 6 metros.

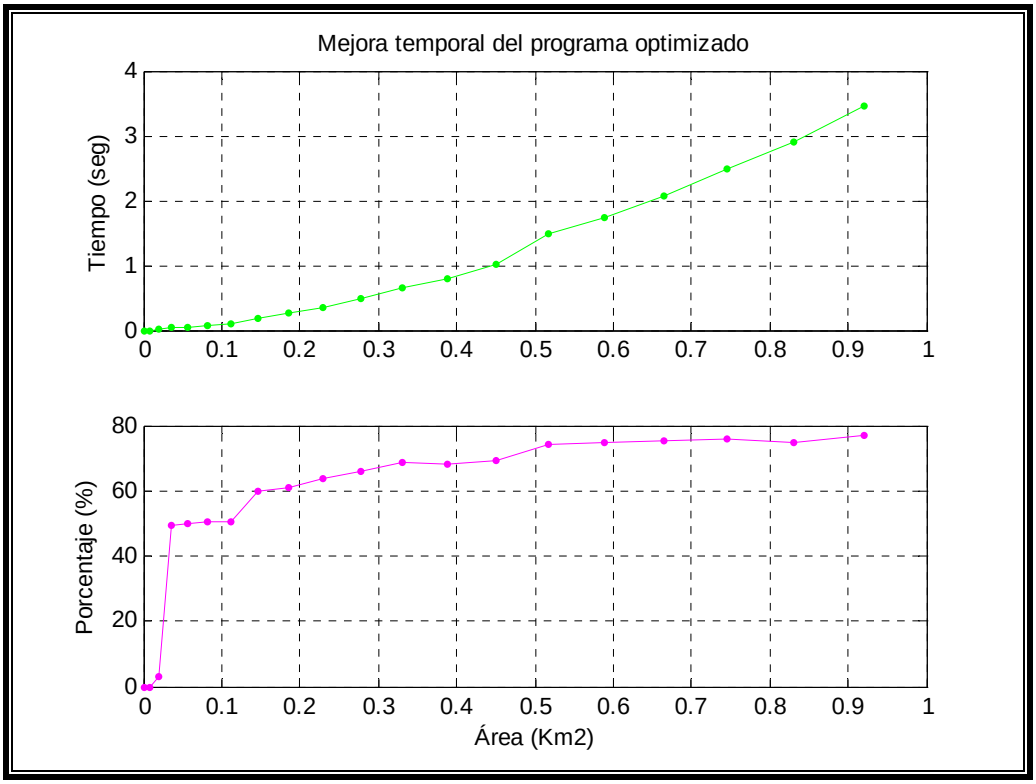


Figura 4.16. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 6 metros.

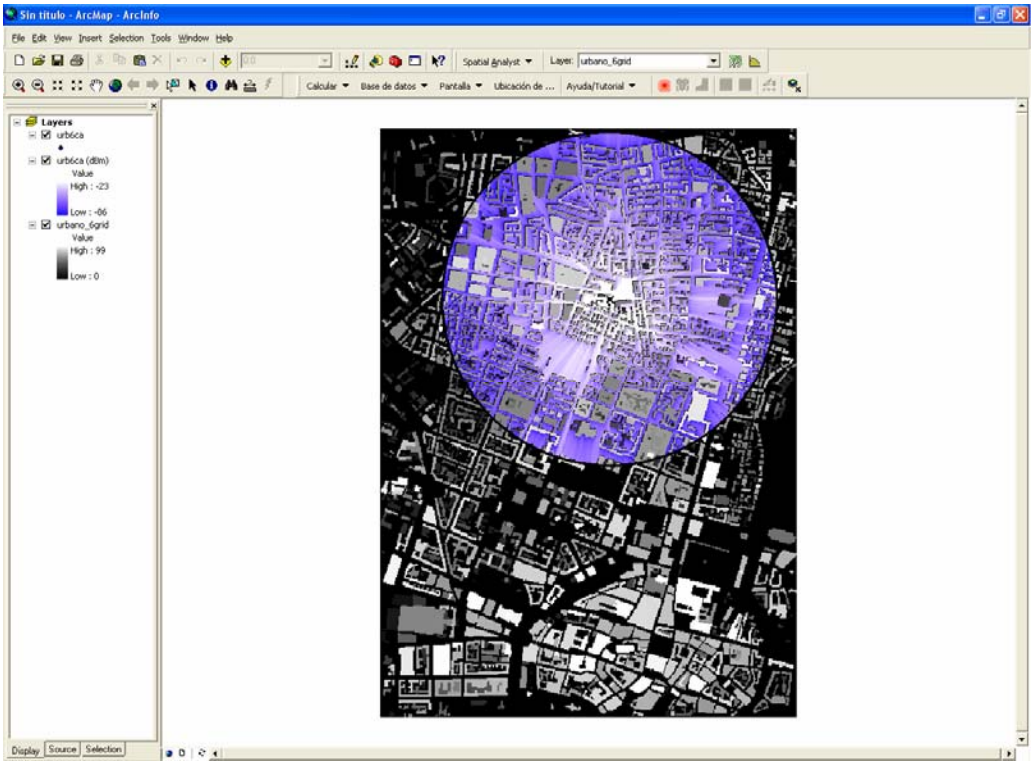


Figura 4.17. Pérdidas calculadas por el programa modificado sobre un área circular, para un MDT urbano de resolución 6 metros.

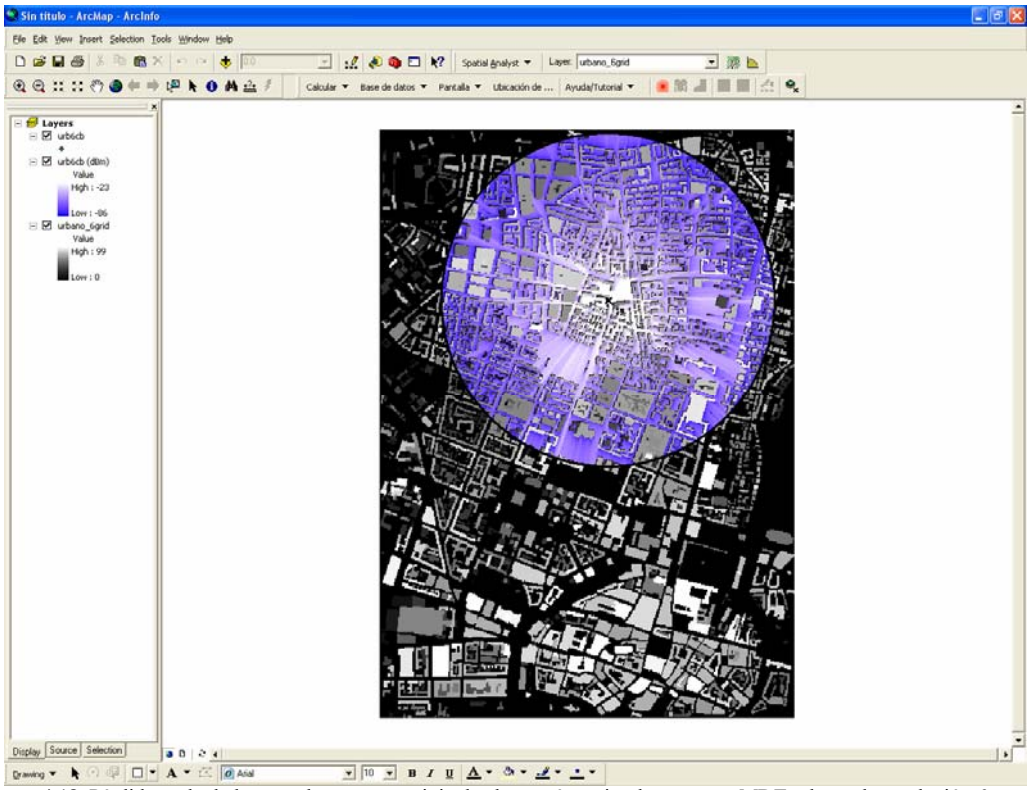


Figura 4.18. Pérdidas calculadas por el programa original sobre un área circular, para un MDT urbano de resolución 6 metros.

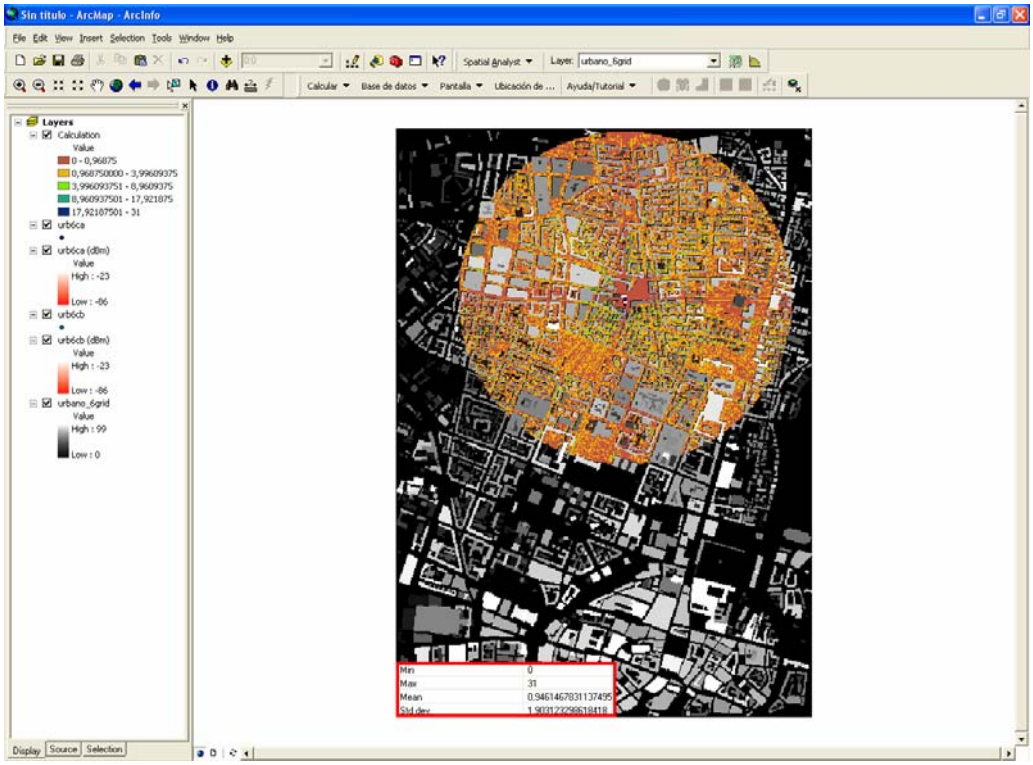


Figura 4.19. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT urbano de resolución 6 metros.

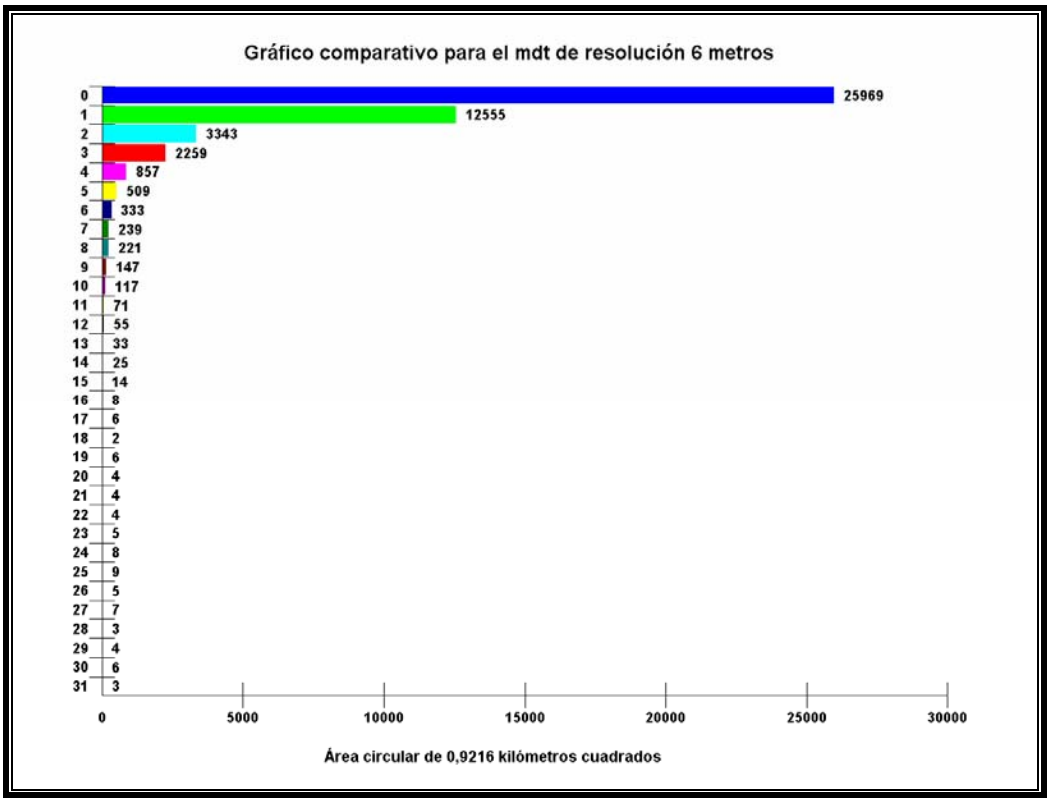


Figura 4.20. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT urbano de resolución 6 metros.

4.5.3. MDT resolución 4.

4.5.3.1. Área rectangular.

MDT4GRID											
LADO (m)	ÁREA (m^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
96	9216	0,015	0,016	0,001	6,250	0	0,0	0	0,0	0	0,0
192	36864	0,031	0,031	0,000	0,000	0	0,0	0	0,0	0	0,0
288	82944	0,047	0,094	0,047	50,000	0	0,0	0	0,1	0	0,0
384	147456	0,063	0,188	0,125	66,489	0	0,1	0	0,2	0	0,1
480	230400	0,109	0,344	0,235	68,314	0	0,1	0	0,3	0	0,2
576	331776	0,172	0,609	0,437	71,757	0	0,2	0	0,6	0	0,4
672	451584	0,250	1,000	0,750	75,000	0	0,3	0	1,0	0	0,8
768	589824	0,328	1,484	1,156	77,898	0	0,3	0	1,5	0	1,2
864	746496	0,438	2,203	1,765	80,118	0	0,4	0	2,2	0	1,8
960	921600	0,562	2,906	2,344	80,661	0	0,6	0	2,9	0	2,3
1056	1115136	0,688	3,844	3,156	82,102	0	0,7	0	3,8	0	3,2
1152	1327104	0,828	4,797	3,969	82,739	0	0,8	0	4,8	0	4,0
1248	1557504	1,016	5,875	4,859	82,706	0	1,0	0	5,9	0	4,9
1344	1806336	1,219	7,172	5,953	83,003	0	1,2	0	7,2	0	6,0
1440	2073600	1,422	8,750	7,328	83,749	0	1,4	0	8,8	0	7,3
1536	2359296	1,672	10,578	8,906	84,194	0	1,7	0	10,6	0	8,9
1632	2663424	1,906	12,609	10,703	84,884	0	1,9	0	12,6	0	10,7
1728	2985984	2,250	15,015	12,765	85,015	0	2,3	0	15,0	0	12,8
1824	3326976	2,546	18,047	15,501	85,892	0	2,5	0	18,0	0	15,5
1920	3686400	2,875	21,469	18,594	86,609	0	2,9	0	21,5	0	18,6

Tabla 4.3. Tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 4 metros.

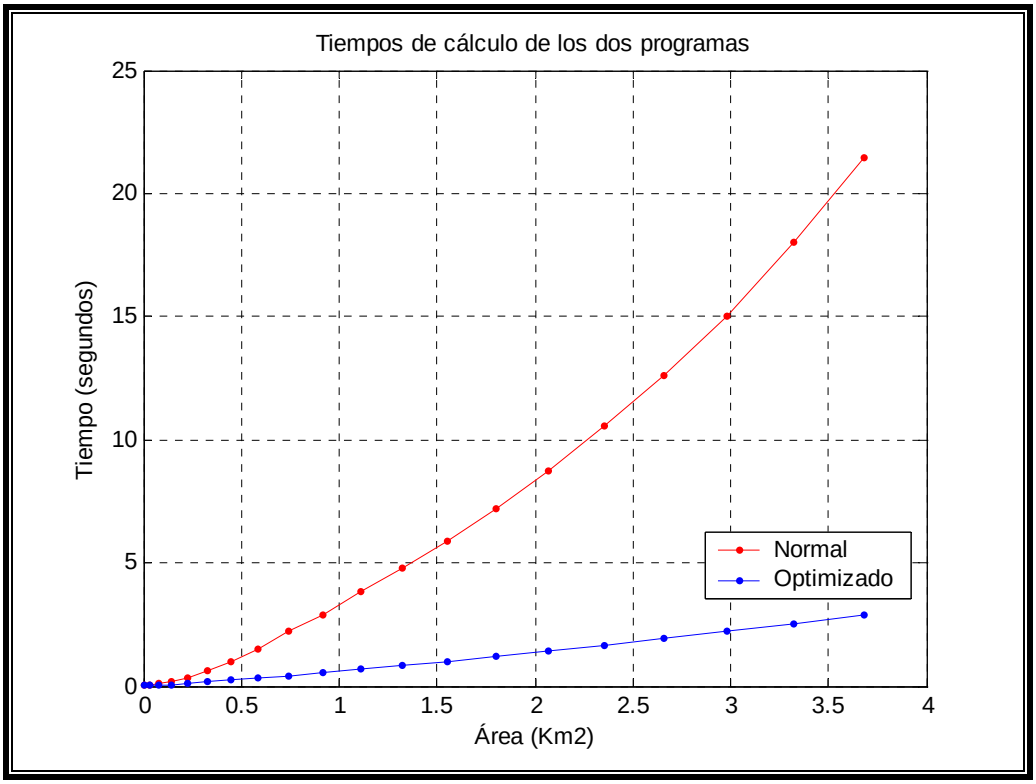


Figura 4.21. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 4 metros.

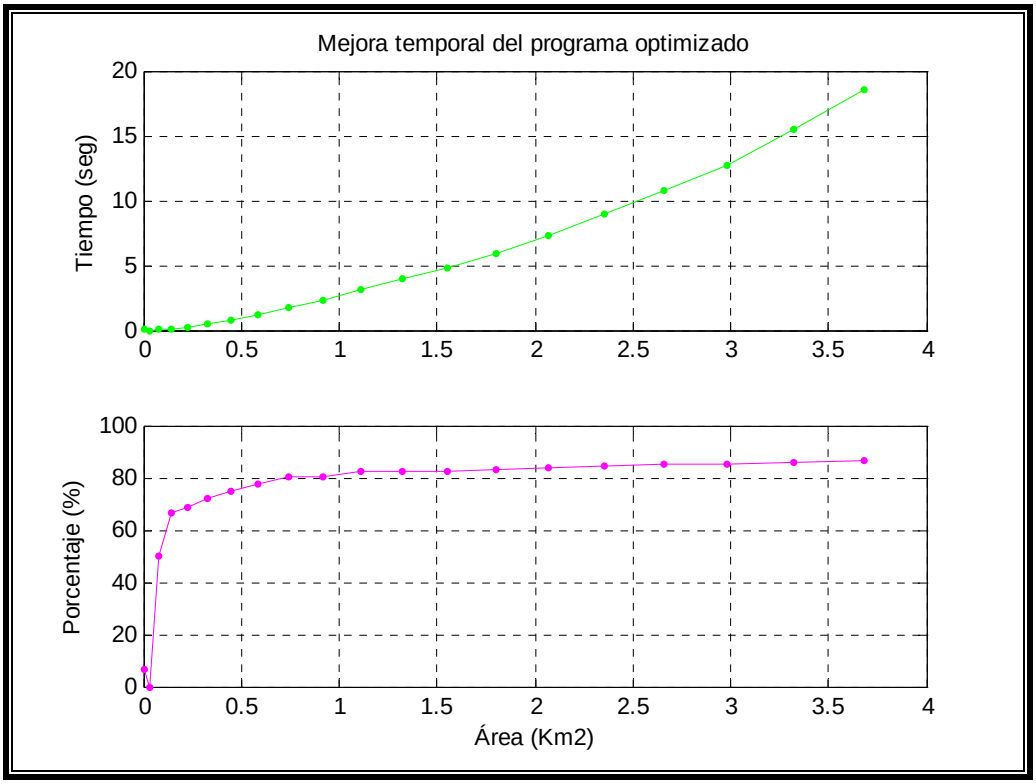


Figura 4.22. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 4 metros.

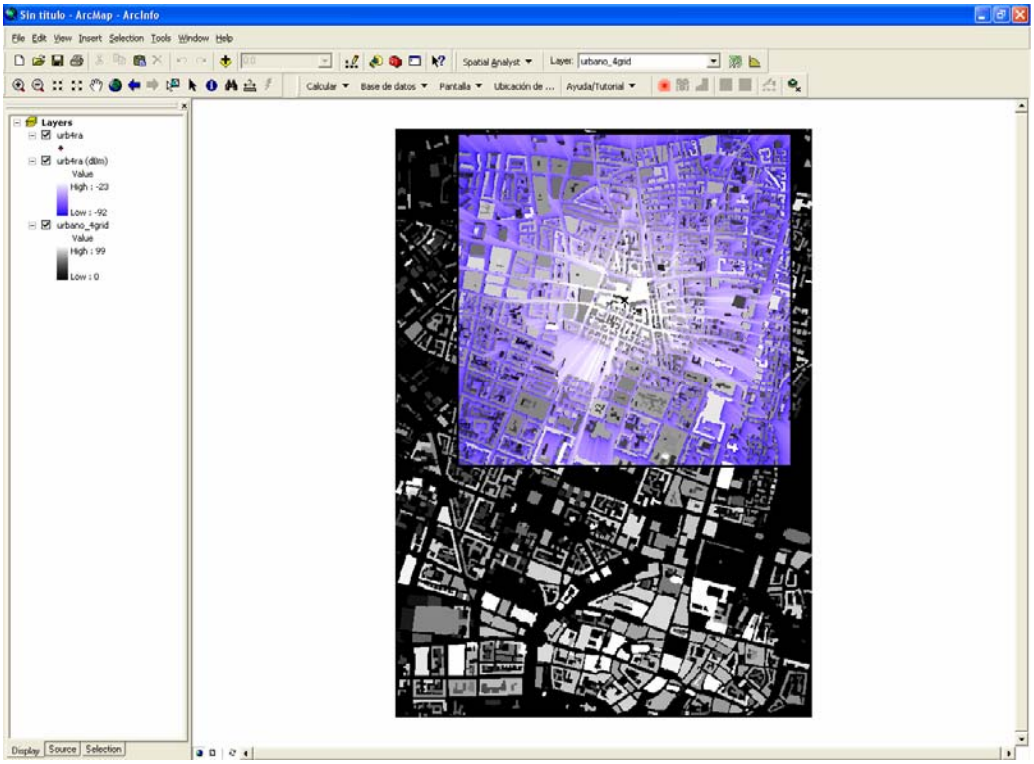


Figura 4.23. Pérdidas calculadas por el programa modificado sobre un área rectangular, para un MDT urbano de resolución 4 metros.

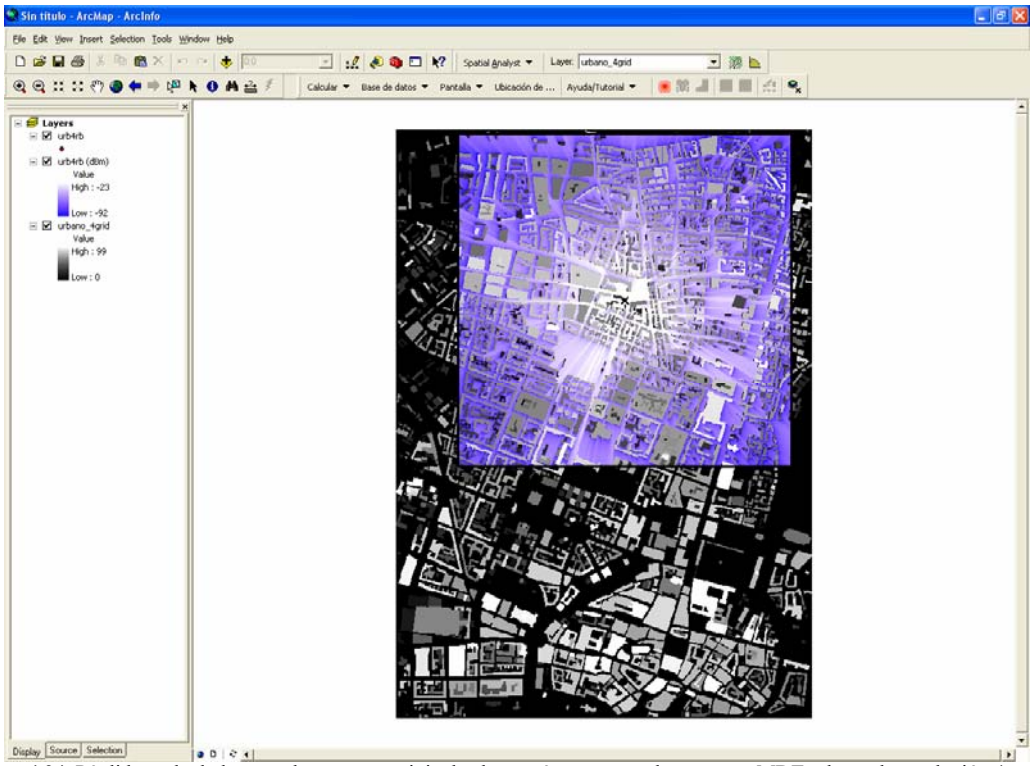


Figura 4.24. Pérdidas calculadas por el programa original sobre un área rectangular, para un MDT urbano de resolución 4 metros.

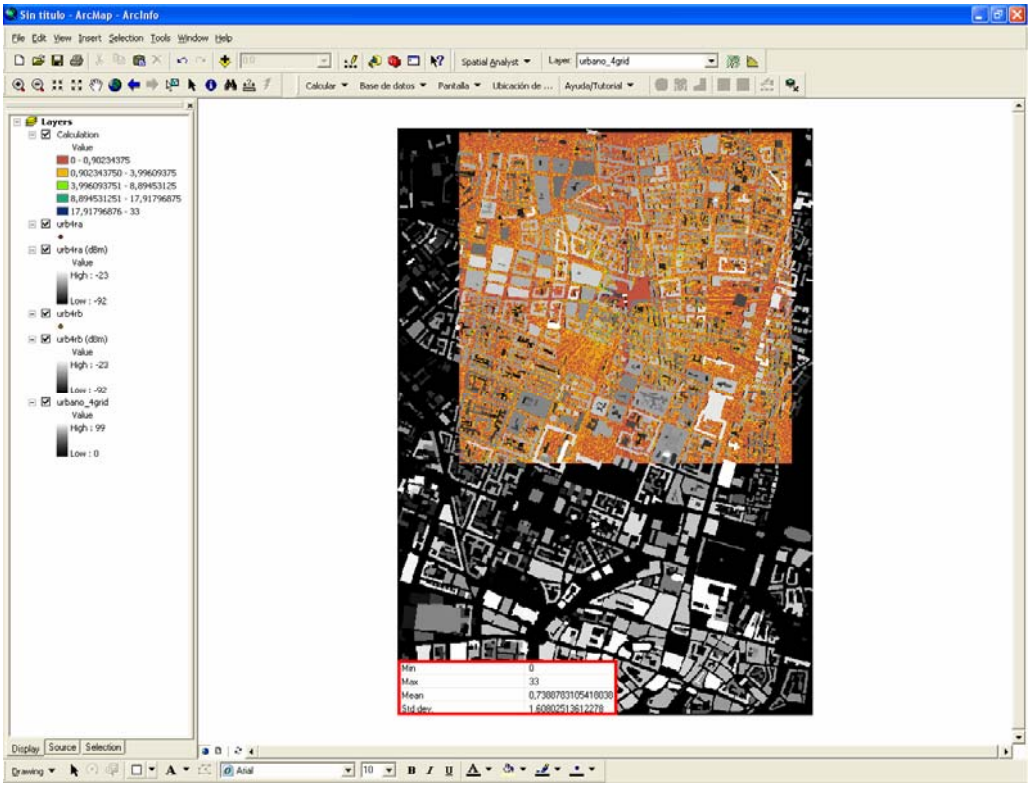


Figura 4.25. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT urbano de resolución 4 metros.

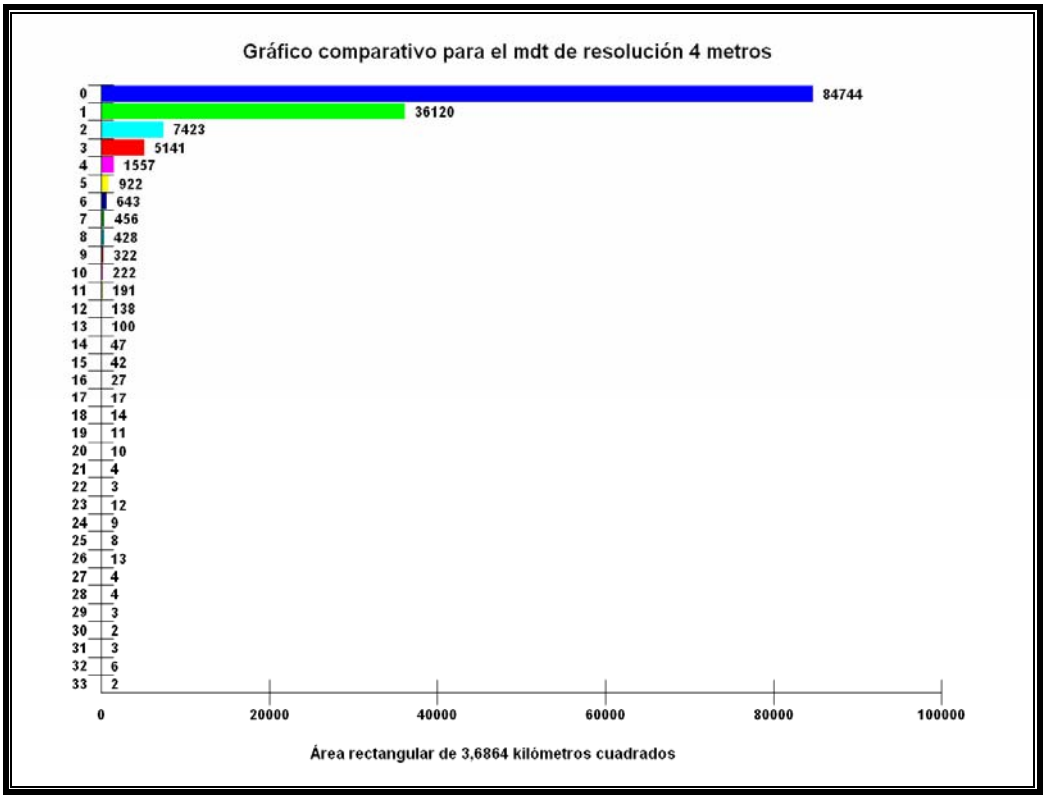


Figura 4.26. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT urbano de resolución 4 metros.

4.5.3.2. Área circular.

MDT4GRID											
RADIO (m)	ÁREA (m^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
48	2304	0,000	0,000	0,000	0,000	0	0,0	0	0,0	0	0,0
96	9216	0,031	0,031	0,000	0,000	0	0,0	0	0,0	0	0,0
144	20736	0,047	0,063	0,016	25,397	0	0,0	0	0,1	0	0,0
192	36864	0,078	0,141	0,063	44,681	0	0,1	0	0,1	0	0,1
240	57600	0,110	0,250	0,140	56,000	0	0,1	0	0,3	0	0,1
288	82944	0,188	0,407	0,219	53,808	0	0,2	0	0,4	0	0,2
336	112896	0,234	0,656	0,422	64,329	0	0,2	0	0,7	0	0,4
384	147456	0,328	1,000	0,672	67,200	0	0,3	0	1,0	0	0,7
432	186624	0,422	1,453	1,031	70,957	0	0,4	0	1,5	0	1,0
480	230400	0,531	1,984	1,453	73,236	0	0,5	0	2,0	0	1,5
528	278784	0,656	2,734	2,078	76,006	0	0,7	0	2,7	0	2,1
576	331776	0,813	3,484	2,671	76,665	0	0,8	0	3,5	0	2,7
624	389376	0,984	4,282	3,298	77,020	0	1,0	0	4,3	0	3,3
672	451584	1,172	5,188	4,016	77,409	0	1,2	0	5,2	0	4,0
720	518400	1,375	6,250	4,875	78,000	0	1,4	0	6,3	0	4,9
768	589824	1,593	7,453	5,860	78,626	0	1,6	0	7,5	0	5,9
816	665856	1,828	8,719	6,891	79,034	0	1,8	0	8,7	0	6,9
864	746496	2,219	10,234	8,015	78,317	0	2,2	0	10,2	0	8,0
912	831744	2,688	12,000	9,312	77,600	0	2,7	0	12,0	0	9,3
960	921600	2,875	14,266	11,391	79,847	0	2,9	0	14,3	0	11,4

Tabla 4.4. Tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 4 metros.

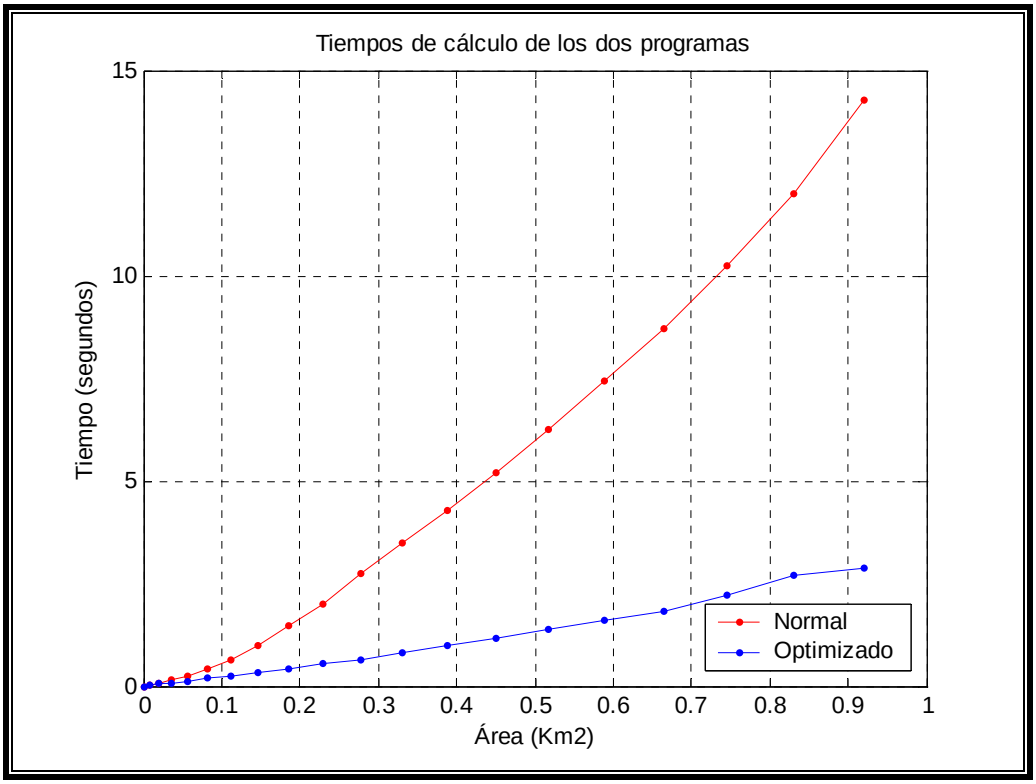


Figura 4.27. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 4 metros.

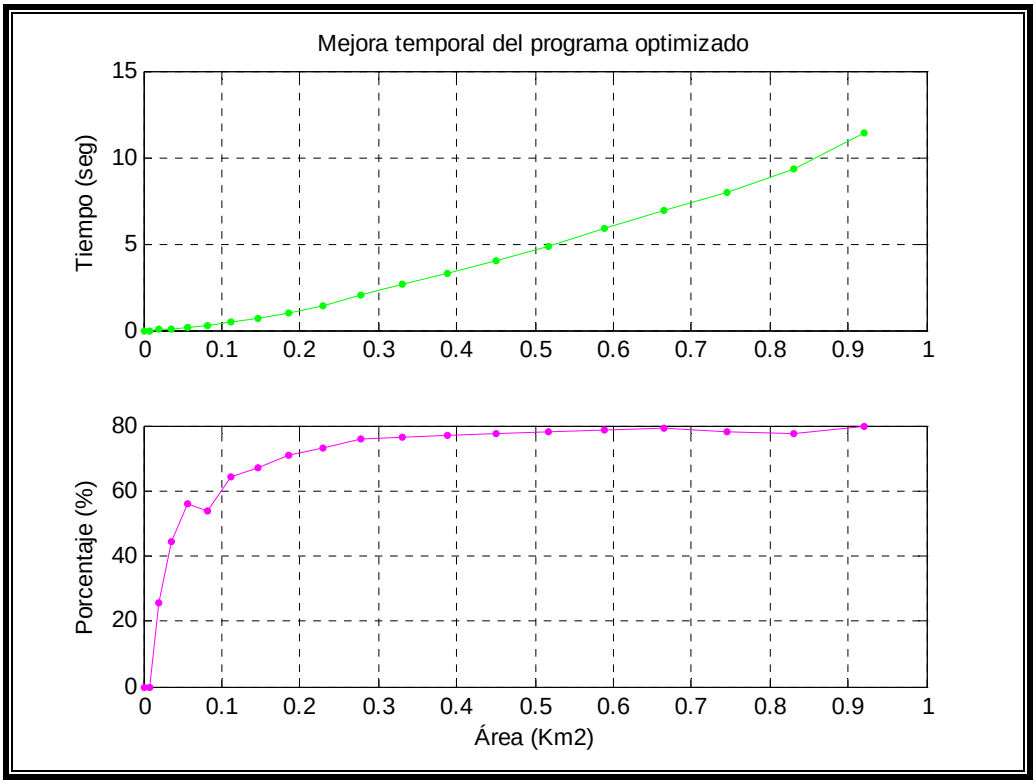


Figura 4.28. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 4 metros.

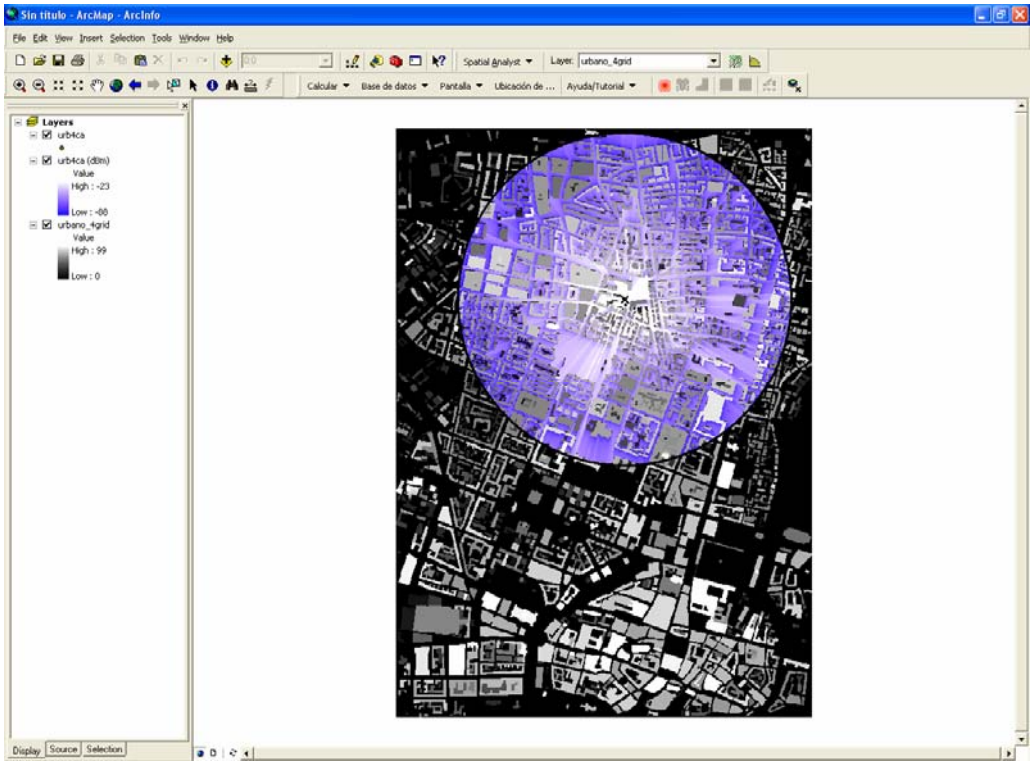


Figura 4.29. Pérdidas calculadas por el programa modificado sobre un área circular, para un MDT urbano de resolución 4 metros.

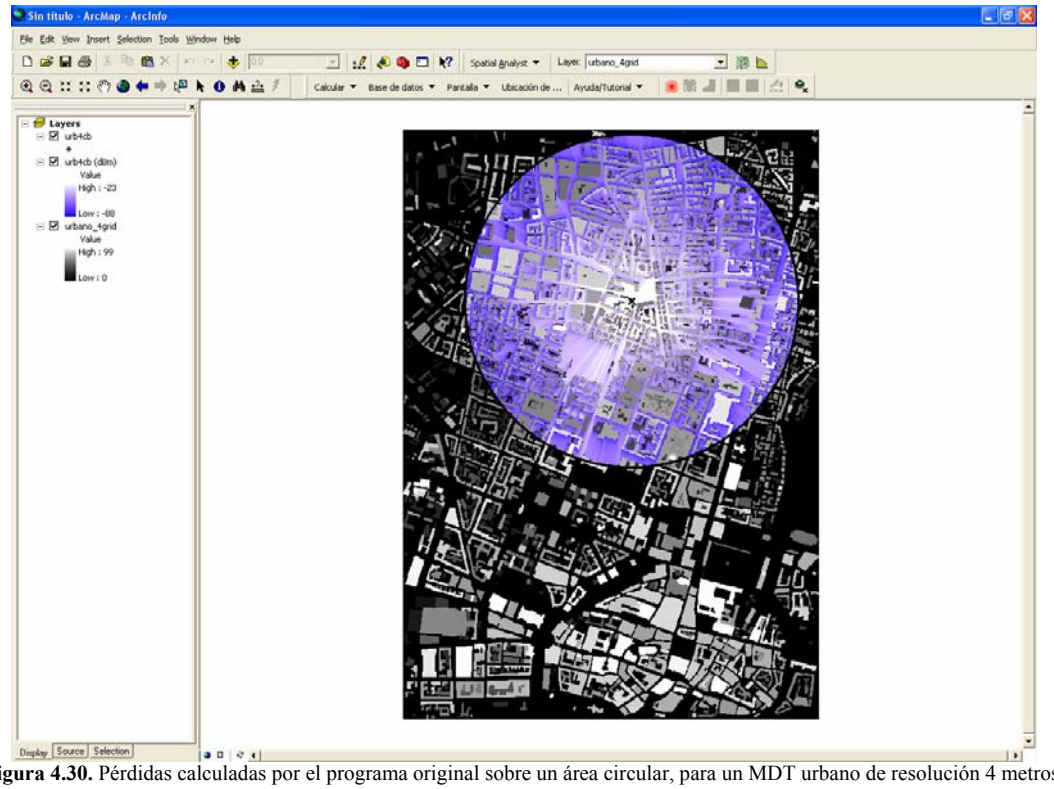


Figura 4.30. Pérdidas calculadas por el programa original sobre un área circular, para un MDT urbano de resolución 4 metros.

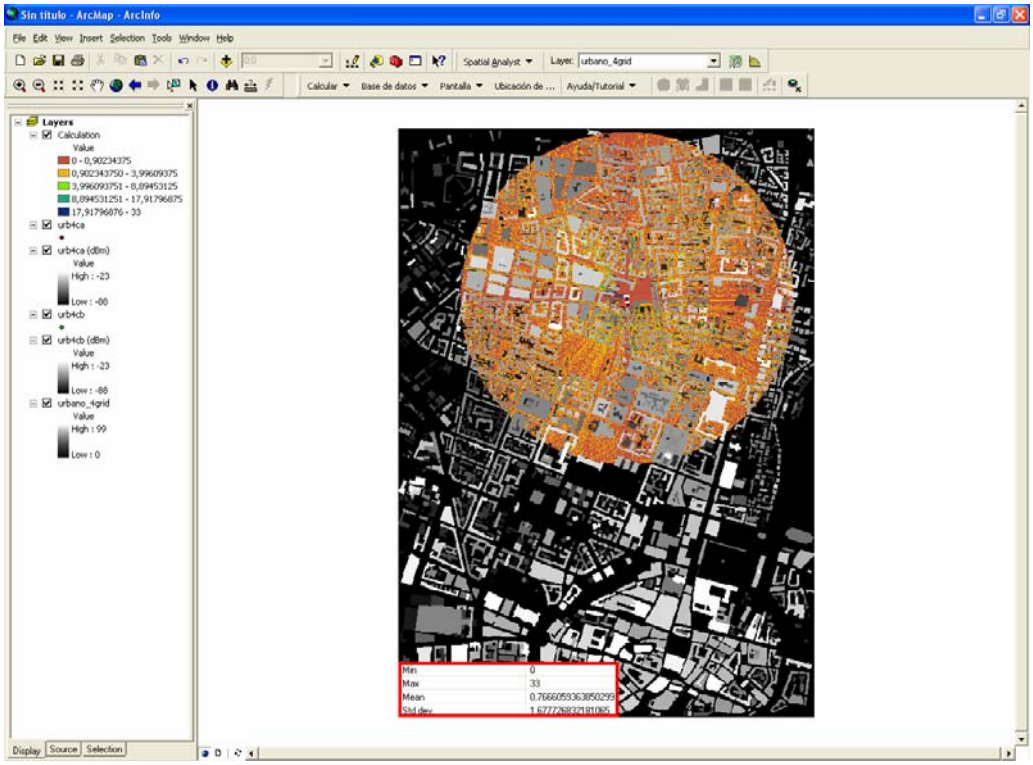


Figura 4.31. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT urbano de resolución 4 metros.

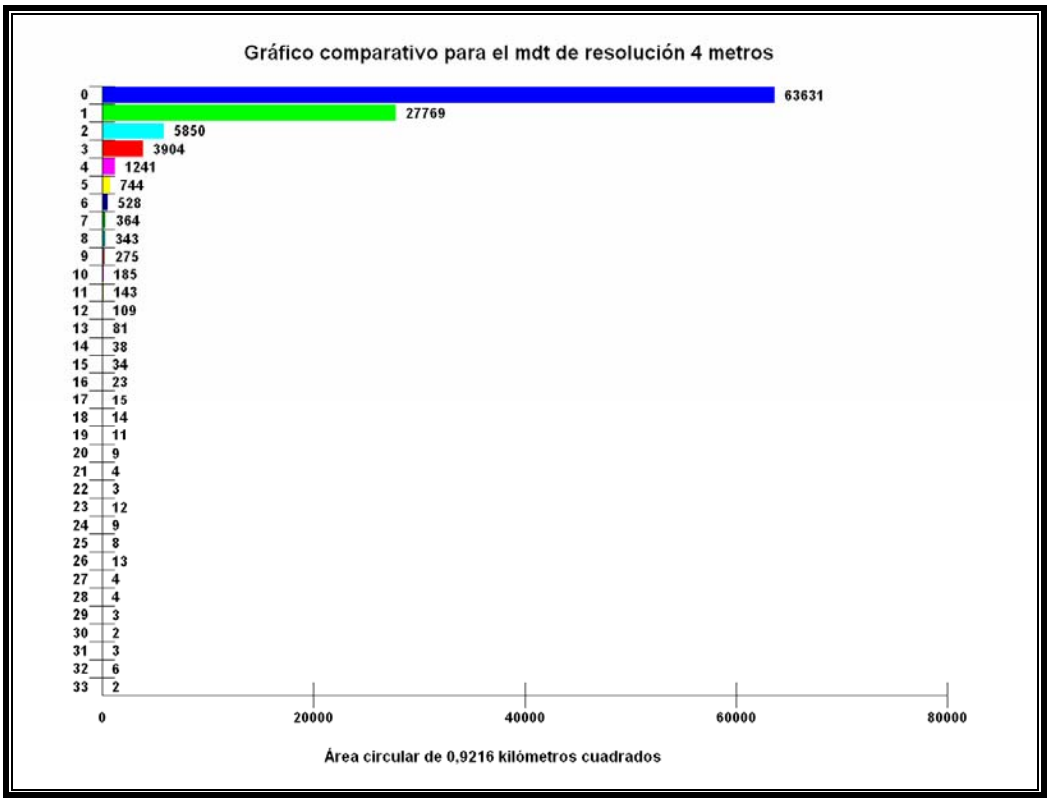


Figura 4.32. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT urbano de resolución 4 metros.

4.5.4. MDT resolución 2.

4.5.4.1. Área rectangular.

MDT2GRID											
LADO (m)	ÁREA (m^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
96	9216	0,016	0,047	0,031	65,957	0	0,0	0	0,0	0	0,0
192	36864	0,062	0,204	0,142	69,608	0	0,1	0	0,2	0	0,1
288	82944	0,157	0,625	0,468	74,880	0	0,2	0	0,6	0	0,5
384	147456	0,297	1,375	1,078	78,400	0	0,3	0	1,4	0	1,1
480	230400	0,516	2,532	2,016	79,621	0	0,5	0	2,5	0	2,0
576	331776	0,781	4,547	3,766	82,824	0	0,8	0	4,5	0	3,8
672	451584	1,141	7,437	6,296	84,658	0	1,1	0	7,4	0	6,3
768	589824	1,578	12,282	10,704	87,152	0	1,6	0	12,3	0	10,7
864	746496	2,109	17,204	15,095	87,741	0	2,1	0	17,2	0	15,1
960	921600	2,704	22,375	19,671	87,915	0	2,7	0	22,4	0	19,7

Tabla 4.5. Tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 2 metros.

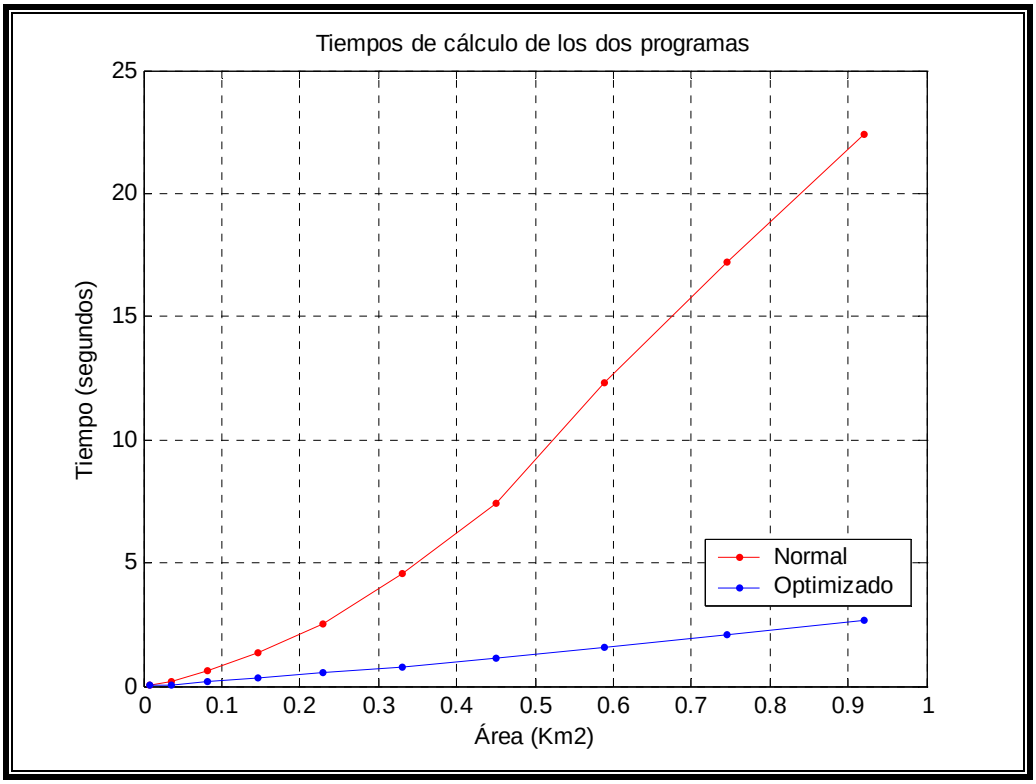


Figura 4.33. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 2 metros.

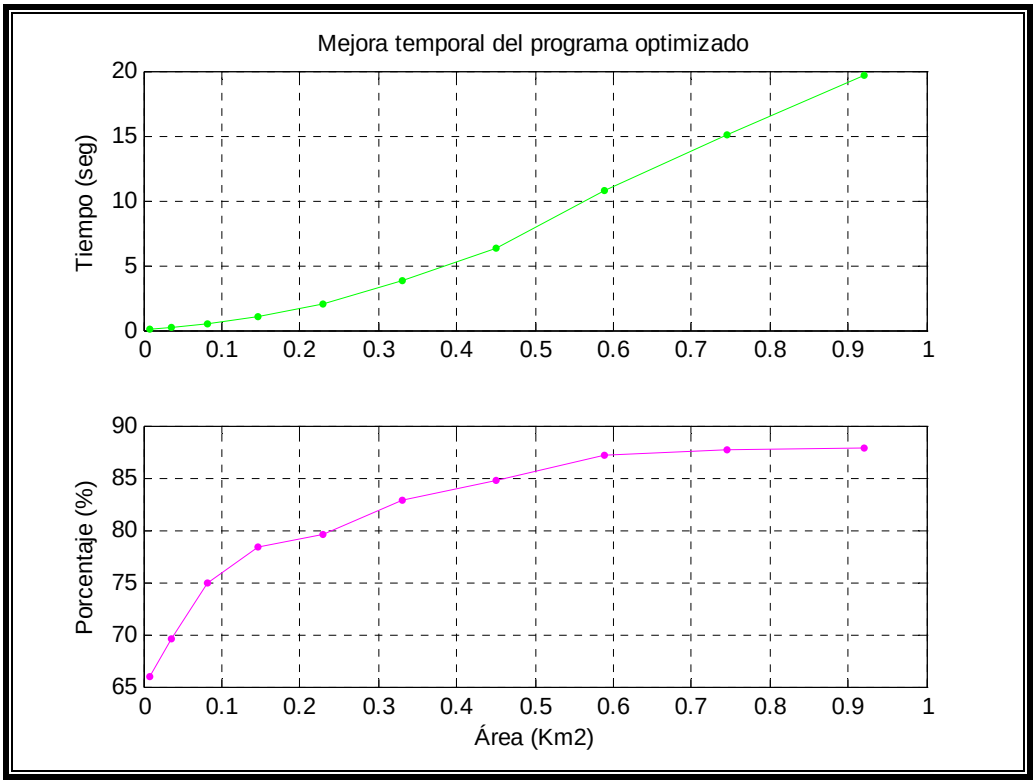


Figura 4.34. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área rectangular, para un MDT urbano de resolución 2 metros.

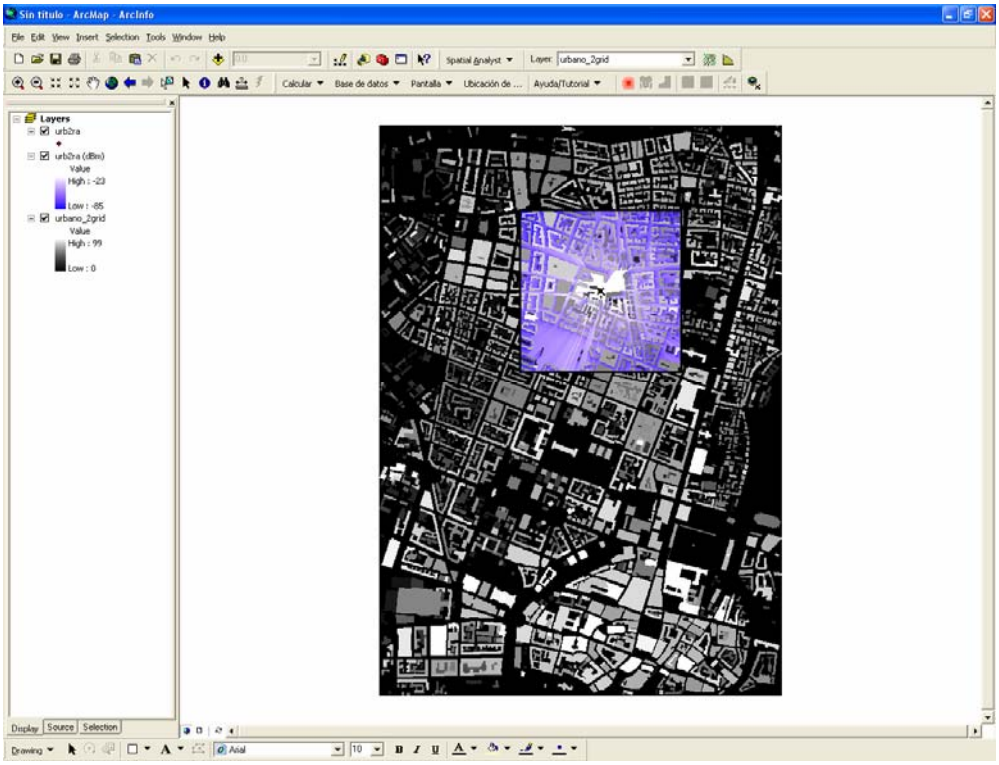


Figura 4.35. Pérdidas calculadas por el programa modificado sobre un área rectangular, para un MDT urbano de resolución 2 metros.

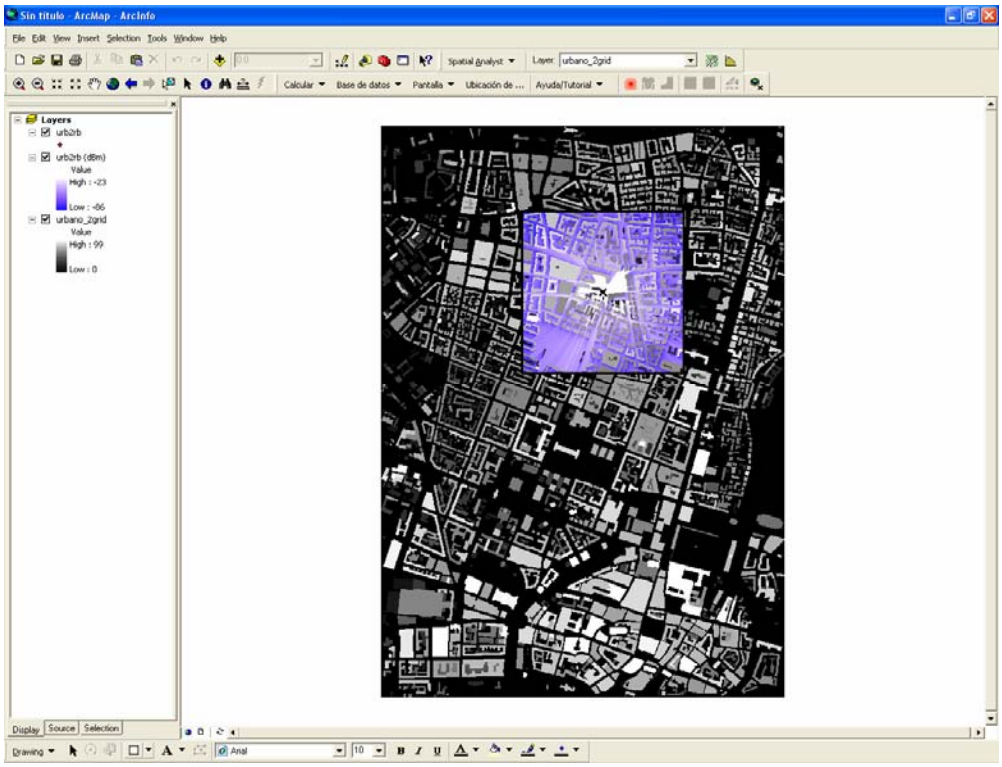


Figura 4.36. Pérdidas calculadas por el programa original sobre un área rectangular, para un MDT urbano de resolución 2 metros.

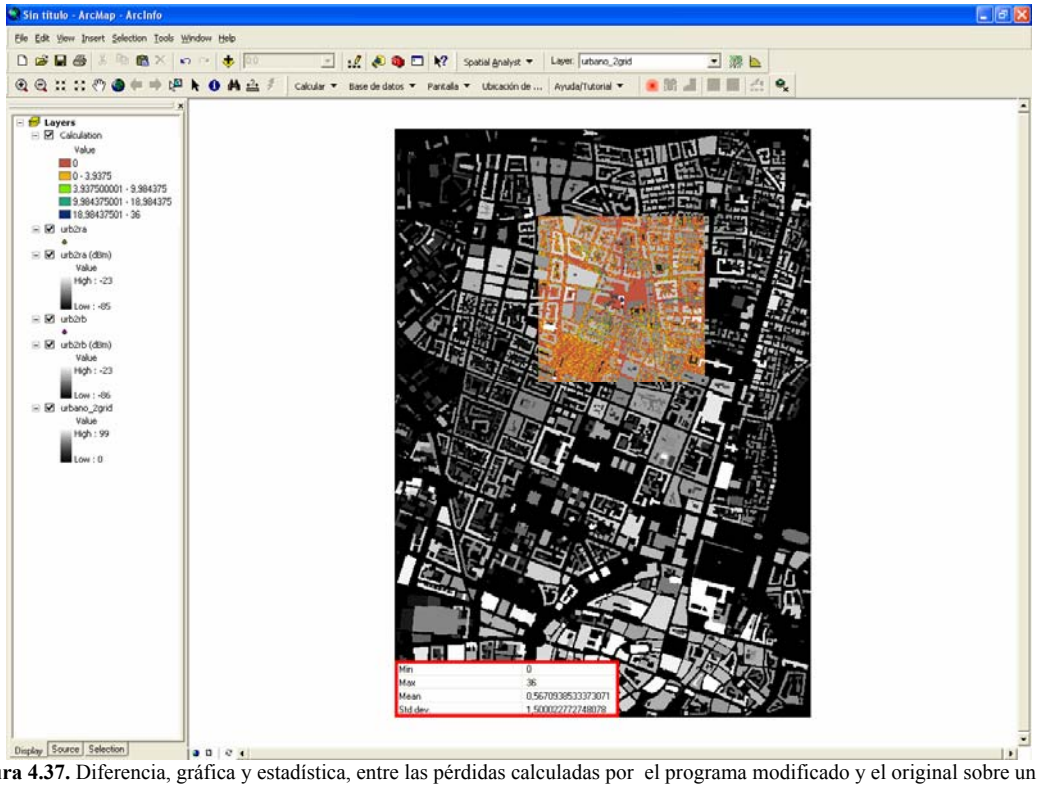


Figura 4.37. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT urbano de resolución 2 metros.

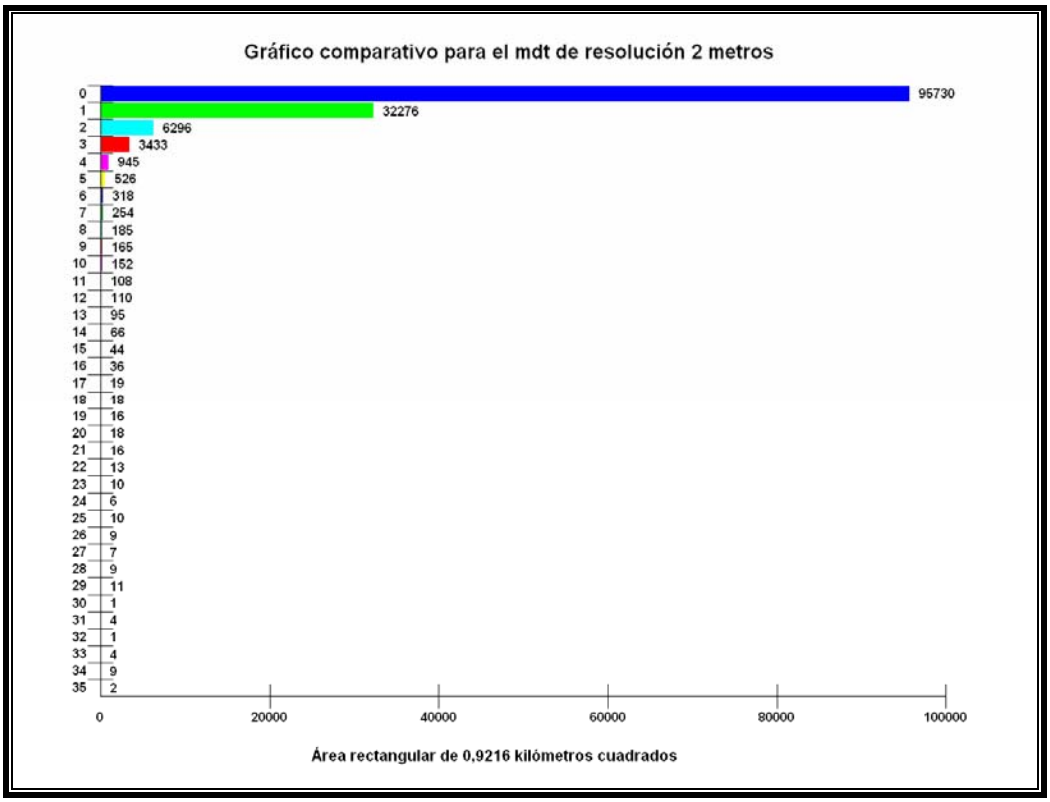


Figura 4.38. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área rectangular, para un MDT urbano de resolución 2 metros.

4.5.4.2. Área circular.

MDT2GRID											
RADIO (m)	ÁREA (m^2)	TIEMPO									
		A (seg)	B (seg)	B-A (seg)	B-A (%)	A		B		B-A	
						min	seg	min	seg	min	seg
48	2304	0,016	0,031	0,015	48,387	0	0,0	0	0,0	0	0,0
96	9216	0,078	0,156	0,078	50,000	0	0,1	0	0,2	0	0,1
144	20736	0,157	0,484	0,327	67,562	0	0,2	0	0,5	0	0,3
192	36864	0,297	1,015	0,718	70,739	0	0,3	0	1,0	0	0,7
240	57600	0,484	1,828	1,344	73,523	0	0,5	0	1,8	0	1,3
288	82944	0,766	3,016	2,250	74,602	0	0,8	0	3,0	0	2,3
336	112896	1,078	4,844	3,766	77,746	0	1,1	0	4,8	0	3,8
384	147456	1,485	7,437	5,952	80,032	0	1,5	0	7,4	0	6,0
432	186624	2,016	10,812	8,796	81,354	0	2,0	0	10,8	0	8,8
480	230400	2,625	14,781	12,156	82,241	0	2,6	0	14,8	0	12,2

Tabla 4.6. Tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 2 metros.

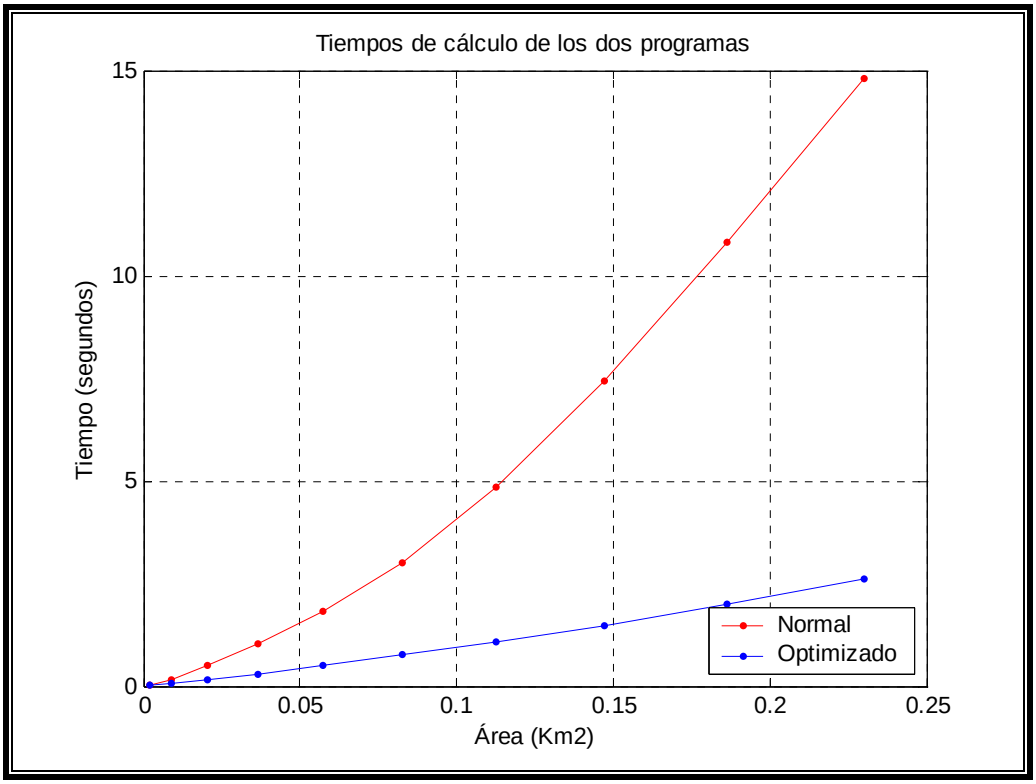


Figura 4.39. Gráfica de los tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 2 metros.

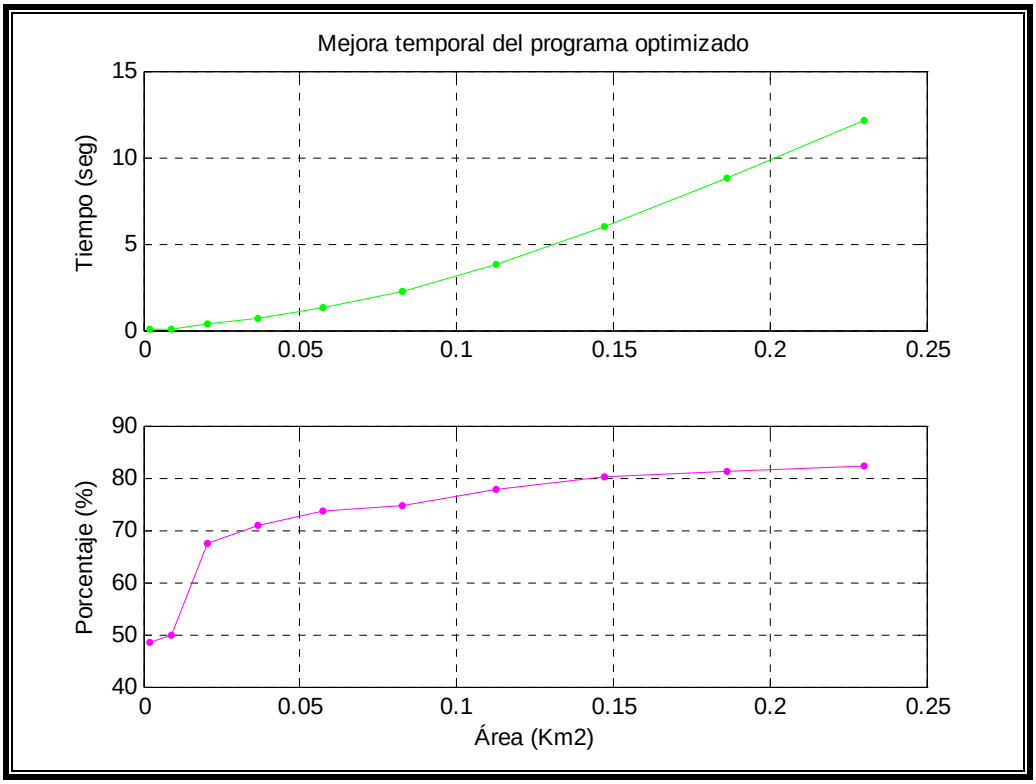


Figura 4.40. Gráficas comparativas de la diferencia de tiempos de cálculo de pérdidas de los programas sobre un área circular, para un MDT urbano de resolución 2 metros.

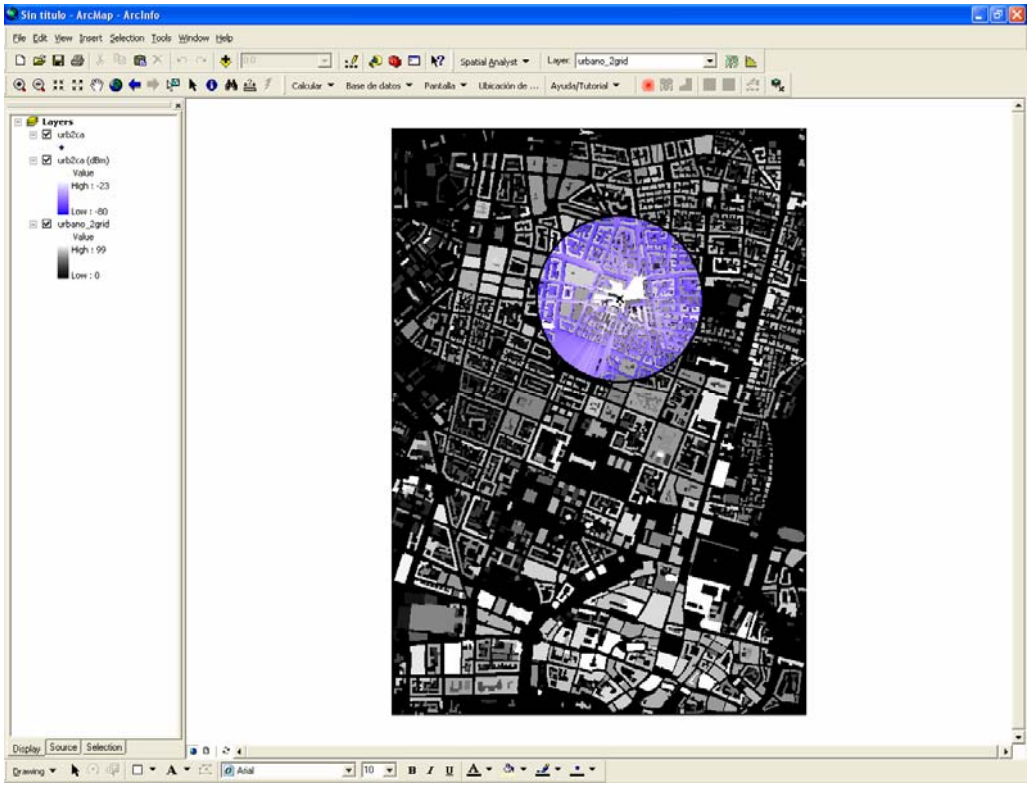


Figura 4.41. Pérdidas calculadas por el programa modificado sobre un área circular, para un MDT urbano de resolución 2 metros.

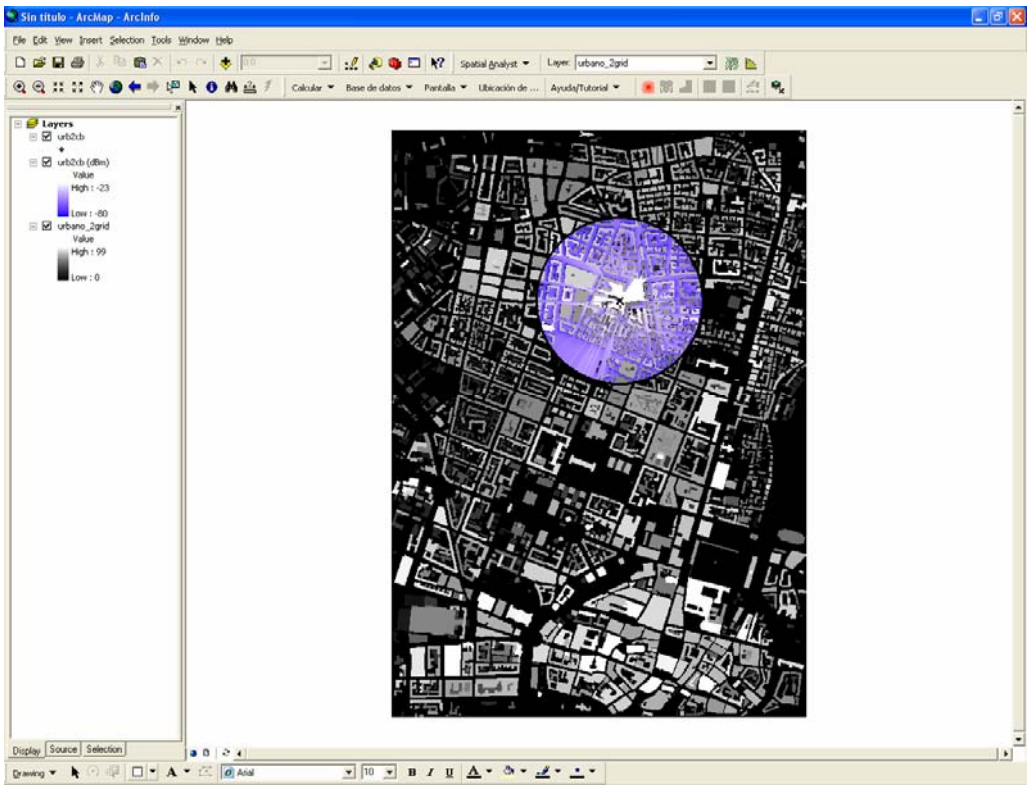


Figura 4.42. Pérdidas calculadas por el programa original sobre un área circular, para un MDT urbano de resolución 2 metros.

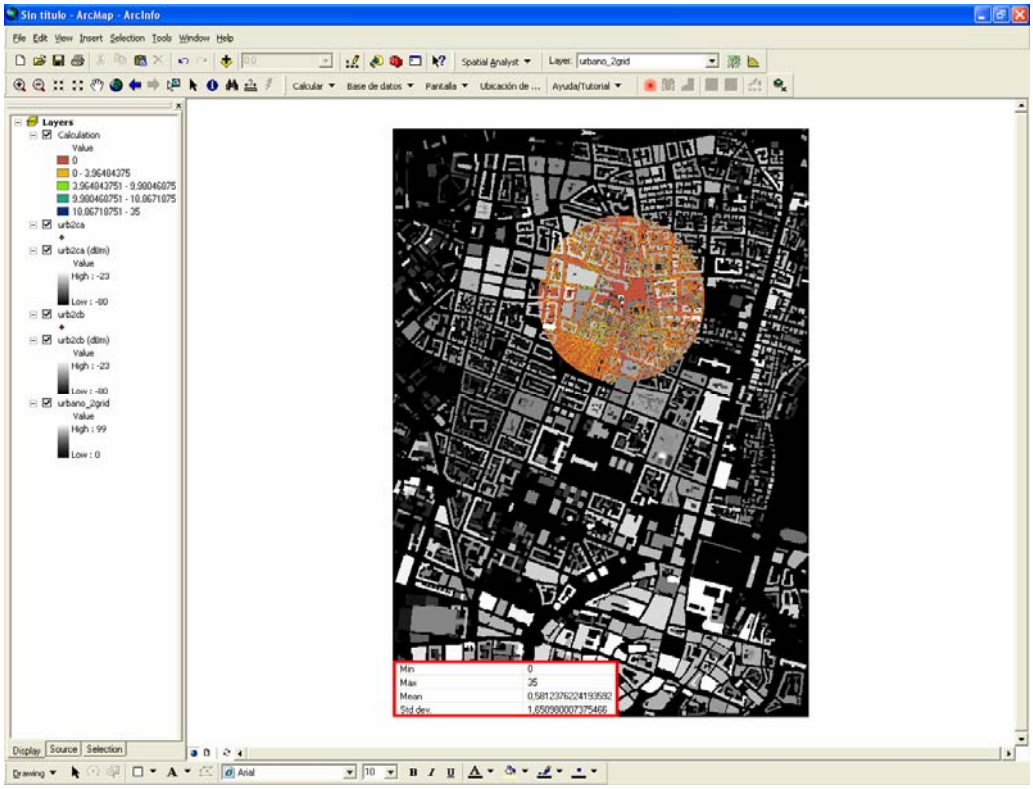


Figura 4.43. Diferencia, gráfica y estadística, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT urbano de resolución 2 metros.

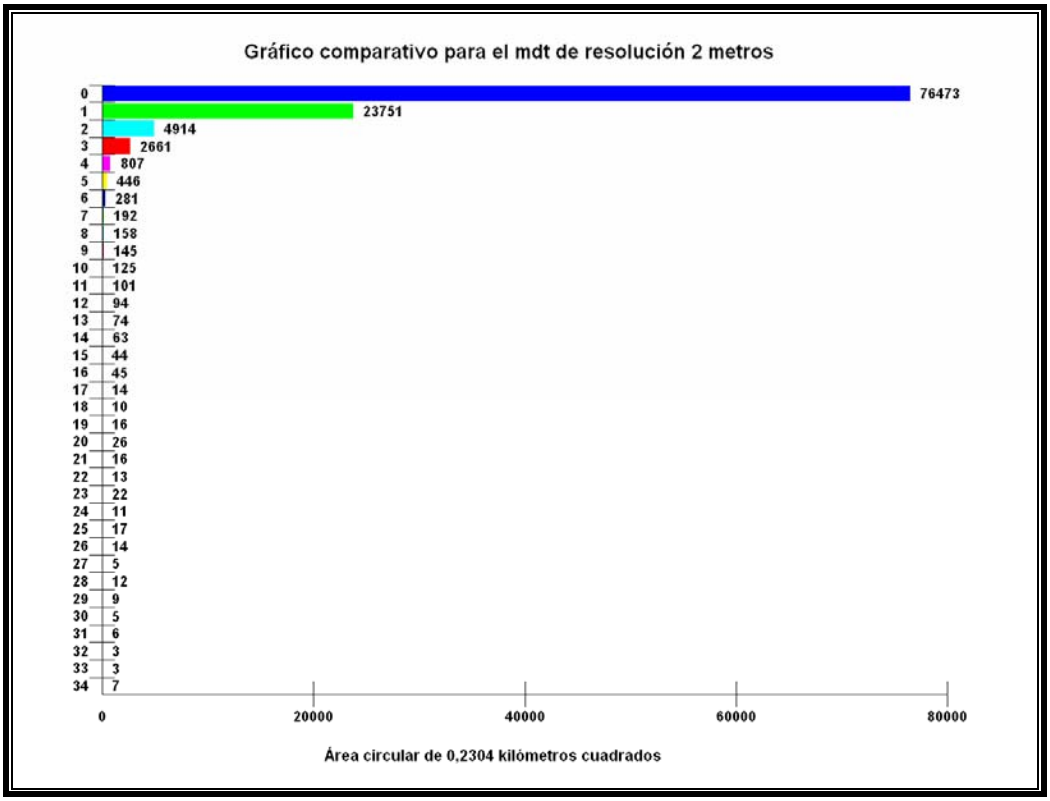


Figura 4.44. Diferencia, relacionados dB's y número de celdas, entre las pérdidas calculadas por el programa modificado y el original sobre un área circular, para un MDT urbano de resolución 2 metros.

CONCLUSIONES Y FUTURA LÍNEA DE TRABAJO

5. CONCLUSIONES Y FUTURA LÍNEA DE TRABAJO.

Una vez desarrollado todo el proyecto, queda concluir si la modificación realizada al código con el que actualmente trabaja *RADIOGIS*, para el cálculo de coberturas de sistemas de radiocomunicaciones en la banda de UHF, es beneficiosa o no.

Lo más adecuado es analizar por separado los resultados obtenidos para los dos factores que darán respuesta a esta cuestión: **tiempo y error de cálculo**.

Antes de indagar al respecto, se va a realizar una mención en relación al número de celdas sobre las que se han calculado las pérdidas.

Como se ha explicado en la tabla 1.1., tanto para el modelo rural como para el urbano, los cálculos se han realizado en mapas digitales de terreno con diferentes resoluciones (distancia real en metros entre celdas contiguas). Esto implica que el número de celdas donde se han calculado las pérdidas es distinto según la resolución; a continuación se puede apreciar dicho número para las simulaciones realizadas sobre el número máximo de celdas consideradas en cada caso.

	RURAL			URBANO		
Resolución (metros)	**200**	**100**	**30**	**6**	**4**	**2**
Nº filas	481	961	1601	320	480	480
Nº columnas	481	961	1601	320	480	480
Nº total de celdas	231361	923521	2563201	102400	230400	230400

Tabla 5.1. Número de celdas del área mayor sobre la que se ha realizado las simulaciones para diferentes resoluciones del MDT.

Ya se sabe que el número de celdas que componen a los diferentes MDTs, es igual ya sea el cálculo para un área circular o rectangular.

1. Ganancia de velocidad de cálculo.

Como se ha comentado en más de una ocasión, la ganancia de tiempo en el cálculo de la cobertura se debe al menor número de ocasiones en las que se levantan

perfiles en el método *main*. En la siguiente tabla se puede observar dicho ahorro, por ejemplo para el modelo rural con área rectangular.

	RURAL - ÁREA RECTANGULAR		
Resolución (metros)	**200**	**100**	**30**
Nº perfiles levantados en el método *main* (fichero original)	231360	923520	2563200
Nº perfiles levantados en el método *main* (fichero modificado)	1920	3840	6400
% ahorro de levantamientos de perfiles en el método *main* tras la modificación	99,170	99,584	99,750

Tabla 5.2. Ahorro de levantamientos de perfiles en el método *main* tras la modificación de código.

El archivo modificado consigue ahorrar más del 99 % de los levantamientos de perfiles realizados por el fichero original en el método *main*, lo cual conlleva a las siguientes optimizaciones temporales.

RURAL	ÁREA RECTANGULAR			ÁREA CIRCULAR		
Resolución (metros)	**200**	**100**	**30**	**200**	**100**	**30**
Nº celdas	231361	923521	2563201	180918	723748	2010567
Diferencia de tiempos en segundos	30,312	248,468	1142,767	19,781	163,610	750,959
Diferencia de tiempos en %	55,491	49,068	47,093	53,327	47,410	48,661
URBANO	ÁREA RECTANGULAR			ÁREA CIRCULAR		
Resolución (metros)	**6**	**4**	**2**	**6**	**4**	**2**
Nº celdas	61624	138648	140924	46831	105384	110585
Diferencia de tiempos en segundos	5,422	18,594	19,671	3,453	11,391	12,156
Diferencia de tiempos en %	83,415	86,609	87,915	77,007	79,847	82,241

Tabla 5.3. Diferencias de tiempos de cálculo en las áreas máximas consideradas para el fichero modificado y el original.

Nota: es interesante remarcar que el número de celdas para las que se realiza el cálculo, es menor en al área circular que en la rectangular.

Durante las simulaciones, teniendo siempre en cuenta el cálculo sobre el número máximo de celdas (áreas mayores) consideradas para cada uno de los MDT de distintas resoluciones, se consigue en el modelo rural un ahorro temporal mínimo del 47,093 % en área rectangular y 47,41 % en circular, y ganancia temporal máxima del 55,491 % para área rectangular y 53,327 % para circular. En el urbano, ese rango oscila entre 77,007 y 82,241 % para área circular, y en área rectangular entre 83,415 y 87, 915 %.

2. Grado de error.

Nota: observar como a continuación, el número de celdas sobre las que se realiza el cálculo, debido a que sobre los edificios no se calculan las pérdidas, no coincide con el total de las que conforman al área de cobertura considerada. Por eso la tabla 5.2. se basa en el MDT rural con área de cálculo rectangular, por ser el único caso en que las pérdidas se hallan para todas las celdas que componen dicha área pasada como argumento (excepto en la celda del transmisor).

	ÁREA RECTANGULAR			ÁREA CIRCULAR		
Resolución (metros)	**200**	**100**	**30**	**200**	**100**	**30**
Nº celdas	231361	923521	2563201	180918	723748	2010567
Diferencia 0 dB's en nº celdas	141218	543310	1661807	114510	441111	1309146
Diferencia 1 dB's en nº celdas	156471	613242	1844749	126250	495084	1448228
Diferencia 2 dB's en nº celdas	166657	658972	1949944	133849	529218	1528793
Diferencia 0 dB's en %	61,04	58,83	64,83	63,29	60,95	65,11
Diferencia 1 dB's en %	67,63	66,40	71,97	69,78	68,41	72,03
Diferencia 2 dB's en %	72,03	71,35	76,07	73,98	73,12	76,04

Tabla 5.4. Diferentes grados de error cometidos en las simulaciones del modelo rural.

En esta tabla se muestra, tanto en número como en tanto por ciento, las celdas sobre las que se ha calculado la potencia de cobertura, y el resultado ha sido con respecto al programa original: igual, diferente en 1 dB, o diferente en 2 dB's (variaciones realmente bajas).

Teniendo en cuenta los diferentes tipos de áreas y resoluciones, como media, en el 62,34 % de los casos los resultados son los mismos; en un 69,37 % difieren como máximo en 1 dB; y la diferencia oscila entre 0 y 2 dB's para el 73,765 %.

Los resultados para el modelo urbano son los siguientes:

	ÁREA RECTANGULAR			ÁREA CIRCULAR		
Resolución (metros)	**6**	**4**	**2**	**6**	**4**	**2**
Nº celdas	61624	138648	140924	46831	105384	110585
Diferencia 0 dB's en nº celdas	34615	84744	95730	25969	63631	76473
Diferencia 1 dB's en nº celdas	51038	120864	128006	38524	91400	100224
Diferencia 2 dB's en nº celdas	55212	128287	134302	41867	97250	105138
Diferencia 0 dB's en %	56,17	61,12	67,93	55,45	60,38	69,15
Diferencia 1 dB's en %	82,82	87,17	90,83	82,26	86,73	90,63
Diferencia 2 dB's en %	89,59	92,53	95,30	89,40	92,28	95,07

Tabla 5.5. Diferentes grados de error cometidos en las simulaciones del modelo urbano.

En este caso, las medias son las siguientes:

- Error inexistente: 61,7 %:
- Error máximo de 1 dB: 86,74 %.
- Error máximo de 2 dB's: 92,36 %.

Para el modelo urbano, el error es prácticamente despreciable. En este caso, la media y desviación típica de error es bastante menor que en el modelo rural (máximas de 0,946 dB's en urbano contra 1,903 dB's en rural). La causa de este hecho se debe a que se trabaja con un menor número de celdas, por lo que la posibilidad de levantar un perfil diferente del que se crearía en el fichero original, es menor.

Concluyendo, **se puede afirmar que se han alcanzado los objetivos del proyecto**.

Una **futura línea** de trabajo debería ser, desde el punto de vista del levantamiento de cada perfil, considerar que el valor de altura para un punto situado a una distancia determinada de la estación base, puede obtenerse como la media espacial de una matriz de nxn celdas alrededor de dicho punto. Se podrían analizar los resultados desde el punto de vista del error cometido y de tiempos de computación.

GLOSARIO

GLOSARIO.

GIS: Sistema de Información Geográfica.

ASCII: Código Estándar Americano para el Intecambio de Información.

MDT: Mapa Digital del Terreno.

TX: Transmisor.

RX: Receptor.

dB´s: debibelios.

UIT: Unión Internacional de Telecomunicaciones.

UHF: Frecuencia Ultra Alta.

FRS: Servicio de Radio Familiar.

PMR446: Radio Móvil Personal a 446 MHz.

NMT: Telefonía Móvil Nórdica.

GSM: Siatema Global de Comunicaciones Móviles.

DCS1800: Sistema Celular Digital a 1800 Mhz.

CCIR: Comité Consultivo Internacional de Radiocomunicaciones.

PIRE: Potencia Isotrópica Radiada Equivalente.

RAM: Memoria de Acceso Aleatorio.

LOS: Línea de Señal con Visión Directa.

NLOS: Línea de Señal sin Vision Directa.

ÍNDICES

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

REFERENCIAS

REFERENCIAS.

1. Byron S. Gottfried (University of Pittsburgh), “Programación en C”. Editorial McGraw-Hill, 1991.

2. J.M. Hernando Rábanos, “Transmisión por radio”. Editorial Centro de Estudios Ramón Areces, 1998.

3. J.M. Hernando Rábanos, “Comunicaciones móviles”. Editorial Centro de Estudios Ramón Areces, 1997.

4. Apuntes de teoría y prácticas de la asignatura “Laboratorio de comunicaciones móviles”, E.T.S. de Ingeniería de Telecomunicación.

5. Tutorial y ayuda de la herramienta ArcGis.

6. www.wikipedia.org.

7. www. catarina.udlap.mx.

8. www. personal.us.es.

9. www. senacitel.cl.