(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 315 428**

(51) Int. Cl.:
G01T 1/164 (2006.01)
G01T 1/29 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Número de solicitud europea: **02797567 .1**
(96) Fecha de presentación : **02.08.2002**
(97) Número de publicación de la solicitud: **1421411**
(97) Fecha de publicación de la solicitud: **26.05.2004**

(54) Título: **Procedimiento para exploraciones SPECT.**

(30) Prioridad: **31.08.2001 DE 101 42 421**

(45) Fecha de publicación de la mención BOPI:
01.04.2009

(45) Fecha de la publicación del folleto de la patente:
01.04.2009

(73) Titular/es: **Forschungszentrum Jülich GmbH**
52425 Jülich, DE
Scivis GmbH, Wissenschaftliche Bildverarbeitung

(72) Inventor/es: **Schramm, Nils;**
Halling, Horst y
Ebel, Gernot

(74) Agente: **González Palmero, Fe**

DESCRIPCIÓN

Procedimiento para exploraciones SPECT.

La invención se refiere a un procedimiento para la tomografía, en particular para la tomografía de cuantos gamma individuales (SPECT).

La tomografía de fotones individuales se refiere a un procedimiento junto con los dispositivos correspondientes para las representaciones tridimensionales de radiofármacos que son introducidos en un objeto. Como objeto pueden estar previstos humanos o animales. Los radiofármacos introducidos en el objeto emiten cuantos gamma. Los cuantos gamma son registrados y evaluados por el dispositivo. Como resultado de la evaluación se obtiene la posición, es decir, la distribución espacial de los radiofármacos en el objeto. La posición de los radiofármacos permite, por su lado, sacar conclusiones relativas al objeto, por ejemplo, de este modo, relativas a una distribución de tejido en el objeto.

Un dispositivo conocido para la realización de una tomografía con cuantos gamma individuales comprende una cámara gamma y un colimador conectado por delante. En el caso del colimador se trata, por reglar general, de una placa de plomo con un gran número de canales que conducen perpendicularmente a través de la placa. Al prever canales se garantiza, por un lado, que sólo se registren los cuantos gamma que inciden perpendicularmente y, por otro lado, que sea posible una medición espacial. La cámara se desplaza, conjuntamente con el colimador, alrededor del objeto. Gracias a ello se obtiene un gran número de informaciones espaciales. En este caso se trata de las denominadas tomas de proyección. A partir de las informaciones espaciales obtenidas alrededor del objeto se puede determinar a continuación la posición de los radiofármacos en el objeto.

Para poder suprimir la radiación difusa ocasionada por los cuantos gamma, se requiere, por lo general, otra información de la energía. Debido a esto, la cámara está realizada, por regla general, de tal manera se puede determinar al mismo tiempo la energía de los cuantos gamma incidentes.

La radiación difusa presenta fundamentalmente una energía menor en comparación con la radiación medida real. De este modo se puede eliminar la radiación difusa no teniendo en cuenta para ello cuantos gamma con menor energía. Igualmente puede ser interesante fijar un límite superior de la energía de los cuantos gamma, para poder eliminar radiación de fondo.

El procedimiento descrito previamente, o bien el dispositivo descrito previamente pertenece al conocimiento técnico general, ya que estos procedimientos y dispositivos se emplean desde hace ya más de treinta años.

La tomografía de cuantos gamma individuales (SPECT) y la tomografía de emisión de positrones (PET) representan instrumentos para la representación cuantitativa de distribuciones de indicadores radioactivos espaciales *in vivo*. Además de en la medicina para personas, estos procedimientos se pueden emplear en la investigación farmacológica y preclínica para el desarrollo y evaluación de nuevas uniones de indicadores radioactivos. Mientras que en el PET hoy en día están disponibles diversos sistemas para la investigación de pequeños animales de laboratorio, en el área del SPECT hasta ahora no ha habido desarrollos correspondientes, o sólo en una dimensión insuficiente, y esto aunque los radiofármacos marcados como TC-99m e I-123 tienen en la medicina nuclear una importancia mayor que los núclidos PET.

Con un SPECT de animales de alta resolución y de alta sensibilidad se conseguiría para la investigación preclínica la ventaja de un procedimiento que cuidaría de los animales, con el que se podrían realizar estudios significativos de modo dinámico y repetible en un individuo. Esto se favorece gracias al hecho de que en los radioisótopos mencionados anteriormente se pueden conseguir actividades extremadamente elevadas (aprox. Factor 100 respecto a núclidos PET), que son indispensables para mediciones sin errores *in vivo* (dosis de masa reducida). Para ello se han de realizar desarrollos que lo acompañan de métodos de marcado correspondientes.

Para mejorar la resolución espacial respecto al estado de la técnica mencionado al comienzo se emplea un colimador de orificio en la tomografía de cuantos gamma individuales. Un colimador de orificio se caracteriza por medio de un orificio único a través del cual penetran los cuantos gamma. En caso de que el objeto se encuentre más cerca del colimador de orificio que de la superficie de una cámara gamma o de un detector, entonces gracias a ello se consigue finalmente una mayor resolución espacial. A través del colimador de orificio penetran los cuantos gamma no exclusivamente de modo perpendicular. En su lugar, éstos entran y vuelven a salir en forma de cono. Puesto que el cono que se encuentra tras el colimador de orificio es mayor que el cono que hay por delante del colimador de orificio, como resultado se consigue una mejora de la resolución espacial en comparación con el estado de la técnica mencionado al comienzo.

En un colimador de orificio se ha de prever una abertura de paso pequeña o un orificio pequeño, a través del cual penetren los cuantos gamma, para conseguir así una buena resolución espacial. Cuando más pequeño es el orificio, sin embargo, menos cuantos gamma entran a través de este orificio. Debido a esto, a medida que se hace más pequeño el orificio decrece, de modo desventajoso, la sensibilidad del dispositivo. La sensibilidad se define como la relación entre la velocidad de cómputo y la actividad existente en el objeto.

Si esta sensibilidad se hace muy pequeña, entonces finalmente ya no es posible la realización de una tomografía de cuantos gamma individuales.

Del documento US 5,245,191 se conoce un procedimiento de tomografía en el que se emplea un colimador multi-orificio. Este documento contiene el aprendizaje de que la resolución es mayor cuanto menor es la distancia entre un objeto y el colimador multi-orificio, o bien cuando mayor se selecciona la distancia entre el detector y el colimador multi-orificio. Sin embargo, del documento se puede extraer que se pueden solapar exclusivamente las proyecciones que inciden en el detector, lo que es descrito como problemático.

El documento "Meikle, Steven y col.: IEEE Transactions on Nuclear Science, Tomo 48, Nº 3, Junio 2001 (2001-06), páginas 816-821, XP011042003 Nueva York, US" describe algunos resultados de una simulación por ordenador con la que se investigan diferentes situaciones de un procedimiento de tomografía usando un colimador multi-orificio y un algoritmo de reconstrucción.

El objetivo de la invención es la creación de un procedimiento del tipo mencionado al comienzo con el que se pueda medir con alta resolución y alta sensibilidad.

El objetivo de la invención se consigue con un procedimiento según la reivindicación 1.

El dispositivo usado para la realización del procedimiento comprende un colimador multi-orificio junto con un detector para el registro de los cuantos gamma que penetran a través del colimador multi-orificio. El colimador, así pues, presenta un gran número de aberturas de paso. En una configuración de la invención, el detector está realizado de tal manera que ésta también es capaz de determinar la energía de los cuantos gamma incidentes.

Puesto que el colimador presenta varios orificios, la sensibilidad del dispositivo aumenta de modo correspondiente. El empleo de un colimador de orificio, frente al empleo del tipo de colimadores con los que sólo se registran los rayos que inciden de modo perpendicular, tiene la ventaja de la mayor resolución espacial. Con ello está disponible un dispositivo con buena resolución espacial y buena sensibilidad.

Durante el funcionamiento del dispositivo, el objeto se encuentra más cerca del colimador multi-orificio que la superficie de la cámara o del detector, para conseguir una buena resolución espacial. En el dispositivo, la sujeción para el objeto (diván del paciente), debido a ello, se encuentra más cerca del colimador multi-orificio que la cámara o el detector.

Las distancias de los diferentes pasos u orificios en el colimador multi-orificio están seleccionadas de tal manera que los conos que inciden sobre la cámara, como máximo, se cortan parcialmente. Para la consecución de una buena resolución espacial, así como una buena sensibilidad, representa una ventaja el hecho de permitir regiones de corte. En una configuración de la invención, éstas no representan más del 30%, preferentemente hasta el 70% de la superficie total de un cono que está formado por medio de los cuantos gamma que penetran a través de un orificio del colimador multi-orificio.

En la tomografía de orificio convencional (tomografía de "pinhole"), el centro del orificio se encuentra en la perpendicular central del detector. Además, el eje del orificio, es decir, el eje de simetría del orificio del colimador, está dispuesto perpendicularmente al detector. Se usa entonces un procedimiento de reconstrucción en el que se parte de que el cuanto gamma, que incide desde el centro del objeto sobre la cámara, conforma un ángulo recto con la superficie de la cámara. La base del cono que se conforma en la cámara es fundamentalmente circular.

Esta situación normalmente no es el caso cuando se emplea un colimador multi-orificio. Debido a esto, se proporciona un procedimiento de reconstrucción que tiene en cuenta las condiciones variables. En caso de que un cuanto gamma originado desde el centro del objeto no incida ya perpendicularmente sobre la superficie de la cámara o del detector, entonces no se conforma ningún cono circular (estado idealizado) en la superficie de la cámara. En su lugar, el cono se conforma fundamentalmente en forma de una elipse en la cámara. Según la invención, este problema se soluciona empleando un procedimiento de reconstrucción iterativo. El punto de partida del procedimiento de reconstrucción iterativo es una distribución asumida en el objeto, y en concreto, por regla general, una distribución espacial. Se calcula entonces qué resultado de medición alcanzaría la distribución asumida. El resultado calculado se compara con el resultado medido real. A continuación se toma una nueva distribución modificada. De nuevo se calcula el resultado conformado en la cámara de esta nueva distribución. Se compara de nuevo. Se constata si la nueva distribución se corresponde mejor con el resultado medido. De esta manera, después de la realización de un número suficiente de pasos, se determina una distribución cuyo resultado calculado coincide suficientemente bien con el resultado real (resultado de medición). El procedimiento de reconstrucción iterativo finaliza, en particular, cuando el resultado calculado coincide con el medido con una precisión prefijada. El procedimiento de iteración comprende, así pues, una denominada proyección hacia delante, es decir, el cálculo del resultado de una distribución asumida.

El procedimiento iterativo presenta además la ventaja de que se pueden calcular regiones de solape de los conos conformados en la superficie de la cámara o del detector, y se pueden comparar con el resultado real. Debido a ello, también por esta razón, se ha de preferir a otros procedimientos de reconstrucción. De este modo, así pues, es posible permitir regiones de solape, y de este modo llegar a buenas resoluciones espaciales.

En una configuración de la invención, el colimador multi-orificio comprende una placa que está hecha de wolframio e iridio. Estos materiales presentan un coeficiente de extinción mejor frente a cuantos gamma en comparación con el plomo. El iridio es el más indicado entre los materiales mencionados para extinguir cuantos gamma. Sin embargo, el iridio es muy caro. Debido a ello, por razones de costes, se emplea wolframio en los lugares en los que los requerimientos relativos al comportamiento de extinción son menores. De iridio se hacen las partes de la placa en las que los requerimientos relativos a la extinción de cuantos gama son especialmente grandes. En este caso se trata, en particular, de las regiones de la placa que limitan con los orificios.

Un orificio en la placa desemboca preferentemente desde ambos lados en forma de embudo en la placa. Aquí son los requerimientos relativos a la extinción especialmente elevados, y en concreto, en particular, en la pared del orificio. Debido a ello, las paredes del embudo están hechas preferentemente de iridio. La placa tiene un grosor entonces típicamente de 3 a 10 mm.

Los cuantos gamma que parten desde el interior del objeto son extinguidos por regla general dependiendo del tejido. Según el estado de la técnica, para tener en cuenta esta extinción en la evaluación se parte de un coeficiente de extinción homogéneo que se corresponde con el coeficiente de extinción del agua. Además, la extinción depende de los contornos del objeto. En una configuración de la invención, en el marco de la evaluación se determina el contorno exterior del objeto, y se calcula la extinción dependiendo del contorno. De esta manera se obtienen resultados mejorados.

La medida para el contorno exterior del objeto es la radiación difusa de Compton. En una configuración de la invención, debido a ello, se mide la radiación difusa de Compton, por ejemplo, en una denominada ventana de Compton. En el procedimiento de reconstrucción se tiene en cuenta la radiación difusa de Compton, y a partir de ella se determina el contorno del objeto.

En caso de que incida un cuanto gamma sobre una cámara o sobre el detector, entonces se mide el lugar de la incidencia con una imprecisión típica de la cámara o del detector. En otra configuración del procedimiento se tiene en cuenta en la proyección hacia delante, en la que se basa el procedimiento de reconstrucción iterativo, la característica de proyección, es decir, la imprecisión típica de la cámara o del detector, en la evaluación. De nuevo, la consideración de la imprecisión en la medición se realiza por medio de un procedimiento de iteración del tipo mencionado anteriormente de modo fiable.

En caso de que una fuente radiante que se encuentre en el objeto se encuentre alejada, en proporción, del colimador multi-orificio (es decir, en una región del objeto que está especialmente alejada del colimador), entonces se reduce la sensibilidad. En una configuración del procedimiento de reconstrucción, en la proyección hacia delante se tiene en cuenta esta sensibilidad decreciente.

El procedimiento de proyección de la cámara o del detector depende igualmente de la distancia que hay entre la fuente radiante y el colimador multi-orificio. Este procedimiento de proyección que varía dependiendo de la distancia se tiene en cuenta igualmente de modo iterativo en una configuración del procedimiento.

A continuación se indican partes de programa adecuadas para un procedimiento de iteración, que son capaces de ejecutar los pasos conformes a la invención mencionados anteriormente. Los programas comprenden los parámetros de entrada mencionados a continuación. Además, se indican los valores típicos de este tipo de parámetros de entrada. El concepto de “pinhole” se usa como sinónimo para el concepto de “oficio” (del colimador multi-orificio).

Por lo que se refiere cálculo del perfil o del contorno del objeto, se lleva a cabo el cálculo del perfil del objeto en una forma de realización en la iteración “cero”. Para ello se hace uso de una característica de la radiación Compton que en realidad degrada la calidad de la imagen: los cuantos gamma detectados en dirección incorrecta.

Los cuantos gamma detectados en la dirección incorrecta representan un sustrato en las proyecciones que empeora de modo sostenido la calidad de la imagen. Sin embargo, también pueden ser útiles. Éstos ocasionan que en prácticamente todos los casos clínicos aparezca toda la extensión del paciente en las proyecciones. Incluso en el caso en el que un indicador radioactivo, es decir un radiofármacos, se haya de acumular de modo muy específico en un órgano definido de un modo muy ajustado, de este modo, sin embargo, también parecen salir cuantos de todas las otras regiones del paciente llevadas a proyección. En realidad, éstos son cuantos que tienen su origen en el órgano definido de un modo ajustado, si bien como consecuencia de la difusión Compton parece que “iluminen” a todo el paciente. Se hace uso de esta circunstancia para calcular el perfil del objeto.

El cálculo se realiza en varias etapas:

1) Realización de proyecciones “binarias”. Representan una simplificación de las proyecciones reales, en tanto que en ellas cada contenido de pixel mayor que cero se pone con el valor uno.

Lo fundamental para ello es la fijación de umbrales realizada por el usuario, que en tanto que sea conclusiva separa la proyección del propio objeto investigado, es decir, del paciente, del fondo. El cálculo del umbral se orienta a un máximo promediado que se conforma a partir de todas las proyecciones.

2) Retroproyección de las proyecciones binarias en el espacio del objeto.

3) A partir de la "variedad" de los vóxeles (pequeño elemento de volumen, en la mayoría de los casos cúbico), es decir, la frecuencia con la que un vóxel se ve bajo la geometría del colimador correspondiente en todos los ángulos de las proyecciones, se fija un umbral (determinado de modo heurístico para la geometría correspondiente, vóxeles que en una primera aproximación pertenecen al espacio del interior del cuerpo. (La limitación del espacio del interior del cuerpo es el perfil del cuerpo.)

4) Plegado múltiple con núcleo de plegado 3d.

5) Repetición del punto 3

6) Ejecución dos veces de:

a) Plegado 3d

b) Adición del vóxel en el que se ha plegado algo al espacio interior del cuerpo.

(Tabla pasa a página siguiente)

Requerimiento de entrada	Significado
Proyecciones (float): maus.prj	Las proyecciones medidas
Anchura de las proyecciones [pix]: 266	Dimensión transversal
Capas en proyección [pix]: 193	Dimensión axial
Número de los ángulos: 60	Número de las proyecciones
Tamaño de píxel en proyección [mm]: 2	Tamaño del píxel de proyección
Alisar proyecciones (no=0/sí=1): 1	Alisado de las proyecciones medidas
Distancia eje de rotación/plano del dibujo [mm]: 155	Distancia del plano del dibujo respecto al eje de rotación
Nombre del modelo de apertura (texto): apertura.txt	Nombre del archivo de apertura (ver más abajo)
Grosor del cristal [mm]: 10	Anchura del escintilador
Absorción del cristal [1/cm]: 1.173	Coeficiente de absorción del cristal
Resolución intrínseca [mm]: 3.3	Resolución intrínseca de la cámara
Intensidad absoluta del gamma [%]: 90	Intensidad absoluta de la línea gamma
Distribución inicial (float): maus.sta	Distribución inicial de la reconstrucción
Anchura de la reconstrucción [pix]: 112	Dimensión transversal
Capas de la reconstrucción [pix]: 262	Dimensión axial
Tamaño del vóxel en la reconstrucción [mm]: 0.4	Tamaño del vóxel del objeto
Número de las iteraciones: 30	Número de iteraciones
Alisar volumen (no=0/sí=1): 1	Alisado de los datos de volumen
¿Factor de representación por vóxel (no=0/sí=1)?: 0	¿Tener en cuenta el PSF por vóxel o plegar con el PSF medio?
Desviación respecto al original (no=0/sí=1)?: 0	¿Determinar desviación respecto al original?
Origen del nombre de la salida:	Origen del nombre de los archivos

```
maus                                    de salida
¿Guardar todos?: 5                      ¿Almacenar                  resultados
                                        intermedios?
```

Construcción del archivo de apertura

Un archivo de apertura tiene típicamente la siguiente construcción:

```
   45     18.435     66.503   1.5   60     22.278       21.53   27.9
   45    -18.435     55.858   1.5   60    -22.278      26.164   27.9
   45     36.871     13.306   1.5   60      39.33     -6.8974   27.9
   45          0     2.6613   1.5   60          0     -1.3859   27.9
   45    -36.871    -7.9839   1.5   60     -39.33      4.1513   27.9
   45     18.435    -50.535   1.5   60     22.278     -28.355   27.9
   45    -18.435    -61.181   1.5   60    -22.278     -23.889   27.9
```

En el caso de la primera entrada se trata del número de los orificios. Cada una de las siguientes filas describe un "pinhole". El significado de las entradas es el siguiente:

Columna	Significado
1	coordenada x del punto central del "pinhole", es decir, distancia del "pinhole" respecto al eje de rotación en mm
2	coordenada y del "pinhole", es decir, desviación transversal del PH
3	coordenada z del "pinhole"; es decir, desviación axial del PH
4	diámetro interior del "pinhole" en forma de embudo doble
5	ángulo de abertura del "pinhole" en grados
6	ángulo de inclinación lateral, es decir, transversal, del eje del "pinhole"
7	ángulo de inclinación axial del eje del PH
8	coeficiente de extinción del material de apertura en 1/cm

```
/*******************************************************
```

```
mpr.c                    Autores:      Dr. Mils Schramm (FZ Jülich)
                                       Dr. Gernot Ebel (Firma Scivis)

                                                          Fecha: 20/08/2001
Variante MLEM de la reconstrucción interactiva multi-"pinhole". El algoritmo tiene en
cuenta la sensibilidad dependiente del lugar y la función de representación dependiente
del lugar de una apertura MP con "pinholes" posicionados/inclinados de modo arbitrario.
*********************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/atat.h>
#include <unistd.h>
#include <math.h>
#include <time.h>

#include <"/home/schramm/src/util/util.h">

#define Wd2 M_PI                                    /* Media región angular */
#define Q                                           /* Resolución del PSF */
#define U 2.5                                       /* Anchura del PSF (98.7%) */

#define sci(m, k, l)   sci[(m)*KL+(k)*L+l]                  /* Proyecciones medidas */
#define prj(l, k)              prj[(l)*k+k]                 /* Proyección calculada */
#define rec(j, i, s)   rec[(j)*NS+(i)+S+s]                  /* Reconstrucción */
#define rot(j, i, s)   rot[(j)*NS+(i)+S+s]                 /* Reconstrucción rotada
*/
#define psf(h, l, k)   psf[(h)*CC+(1+Cd2)*C+k+Cd2]      /* Factor de representación */
#define kn2(l, k)              kn2[(1+Ed2)*E+k+Ed2]                 /* Núcleo 2D */
#define kn3(l, k, h)   kn3[(1+Ed2)*EE+k+eD2)*E+h+Ed2]               /* Núcleo 3D */

int N, S, M, K, L, NN, NS, KL, T, C, CC, CD2;
float P, A, q, p, W, cth, cms, gin, res;
float *x2, *y2, *z2, *hd, *alf, *psi, *the, *ams;

float   *PSF( float *sen, unsigned char *ige, unsigned char *ihb );
```

```
void    SUM( float *prj, float *rot, float *sen, unsigned char *ise, float *psf,
           unsigned char *ihb );
void    PRJ( float *prj, float *rot, float *sen, unsigned char *ise, float *psf,
           unsigned char *ihb );
void    QUO( float *prj, float *sci, int m );
void    REP( float *rot, float *ron, float *prj, float *sen, unsigned char *ise );
void    NRM( float *cor, float *nrm );
void    COR( float *rec, float *cor);
void    SAV( float *rec, float *rot, char *basen, int iter );
void    ROT( float *rot, float *rec, float pbi );
void    RAY( float *V, float *N, float *a, float *b, float *o, float *d, float r, float
           t, float i2mico );
void    PIX( float *F, float *a, float *b, float *c, float *d, float r, float t, float
           i2mico, float ta, float ct );
void    SM2( float *sci, float *prj );
void    SM3( float *rec, float *rot );
float   LLK( float *prj, float *sci, int m );
void    DEV( FILE *devf. float *obj, float *rec, float llb, int c );

/*************************************************************************
MAIN(): Programa principal con entradas de usuario. Inicialización de la memoria y del
       propio bucle de recostrucción
*************************************************************************/

main()
{
       unsigned char *ise, *ihb;
       char scin[256], aptn[256], stan[256], objn[26], basen[256], fulln[256];
       int s, i, j, m, k, l, t, stat, ticks, aflag, dflag, pflag, vflag, I, nsave;
       float *sci, *prj, *rec, *rot, *cor, *nrm, *ron, *sen, *psf, *obj, llh;
       FILE *part, *aptf, *devf;

/*====== 0) Entradas del usuario ======================*/

       printf( "\nProyecciones (float): " );
```

```
scanf( "%s", scin );
printf( "Anchura de las proyecciones [pix]: " );
scanf( "%i", &L );
printf( "Capas en proyección [pix]: " );
scanf( "%i", &K );
printf( "Número de los ángulos: " );
scanf( "%i", &M );
printf( "Tamaño de los píxeles en la proyección [mm]: " );
scanf( "%g", &P );
printf( "Proyecciones lisas (no=0 / sí=1)?: " );
scanf( "%i", &pflag );
printf( "Distancia entre eje de rotación / plano del dibujo [mm]: " );
scanf( "%g", &A );
printf( "Nombre del patrón de apertura [text]: " );
scanf( "%s", &aptn );
printf( "\n" );

printf( "Grosor del cristal (mm): " );
scanf("%g ", &cth );
printf( "Absorción del cristal (1/cm): " );
scanf("%g ", &cmu );
printf( " Resolución intrínsica (mm): " );
scanf("%g ", &res );
printf( " Intensidad absoluta del gama [%]: ");
scanf("%g ", &gin );
printf( "\n" );

printf( "Distribución inicial (float): " );
scanf( "%s", stan );
printf( "Anchura de la recon. [pix]: " );
scanf( "%i", &N );
printf( "Capas de la recon. [pix]: " );
scanf( "%i", &S );
printf( "Tamaño de voxel en recon. [mm]: " );
scanf( "%g", &p );
```

```
        printf( "Número de las iteraciones: " );
        scanf( "%i", &I );
        printf( "Volumen alisado (no=0/sí=1): " );
        scanf( "%i", &vflag );
        printf( "Factor de representación por voxel (no=0/sí=1): " );
        scanf( "%i", &aflag );
        printf( "\n" );

        printf( "Desviación respecto al original (no=0/sí=1)?: " );
        scanf( "%i", &dflag );
        if (dflag == 1 )
        {
                printf ( "Datos originales (float): " );
                scanf( "%s", objn );
        }
        printf ( "Nombre original de la salida: " );
        scanf( "%s", basen );
        printf( "%i", &nsave );
        printf( "\n" );
/*====== 1) Guardar parámetros ==============================*/

        sprintf( fulln, "%s.par", basen );
        parf = FOPEN( fulln, "wt" );

        fprintf( parf, "\nAlgoritmo: mpr.c\n" );
        fprintf( parf, "Archivo de parámetros (texto): %s.par\n", basen );
        fprintf( parf, "\n );

        fprintf( parf, "Proyecciones (float): %s\n", scin );
        fprintf( parf, "Anchura de la proyección [pix]: %i\n*, L );
        fprintf( parf, "Capas en proyección [pix]: %i\n", K );
        fprintf( parf, "Número de los ángulos: %i\n", M );
        fprintf( parf, "Tamaño de pixel en proyección [mm]: %g\n", P );
        fprintf( parf, "Proyecciones alisado (no=0/sí=1): %i\n", pflag );
        fprintf( parf, "Distancia plano del dibujo / eje de rotación [mm]: %g\n", A );
```

```
        fprintf( parf, "Nombre del patrón de apertura (texto): %s\n", aptn );
        fprintf( parf, "\n" );

        fprintf( parf, "Grosor del cristal [mm]: %g\n", eth );
        fprintf( parf, "Absorción del cristal [1/cm]: %g\n", cmu );
        fprintf( parf, "Resolución intrínsica [mm]: %g\n", res );
        fprintf( parf, "Intensidad absoluta del gamma [%]: %g\n", gin );
        fprintf( parf, "\n" );

        fprintf( "Distribución inicial (float): %s\n", stan );
        fprintf( "Anchura de la recon. [pix]: %i\n", N );
        fprintf( "Capas de la recon. [pix]: %i\n", S );
        fprintf( "Tamaño de voxel en recon. [mm]: %g\n", p );
        fprintf( "Número de las iteraciones: %i\n", I );
        fprintf( "Volumen alisado (no=0/sí=1): %i\n", vflag );
        fprintf( "Factor de representación por voxel (no=0/sí=1)?: %i\n", aflag );
        fprintf( "\n" );

        fprintf(parf, "Desviación respecto al original (no=0/sí=1)?: %i\n", dflag );
        if (dflag == 1 )
        {
                fprintf ( "Datos originales (float): %s\n", objn );
                fprintf( parf, "Archivo de desviación (texto): %s.dev\n", basen );
        }
        fprintf( parf, "Reconstrucción: %s.rnn\n", basen );
        fprintf( parf, "¿Guardar todos?: %i\n", nsave);
        fprintf( parf, "\n" );

        FCLOSE ( parf );
/*====== 2) Almacenar patrón de apertura ==============================*/

        fprintf( stderr, "Inicializando...\n" );

        aptf = FOPEN( aptn, "rt" );
        fscanf( aptf, "%i", &T );
```

```
        x2 = MALLOC( T*sizeof( float ) );
        y2 = MALLOC( T*sizeof( float ) );
        z2 = MALLOC( T*sizeof( float ) );
        hd = MALLOC( T*sizeof( float ) );
        alf = MALLOC( T*sizeof( float ) );
        psi = MALLOC( T*sizeof( float ) );
        the = MALLOC( T*sizeof( float ) );
        amu = MALLOC( T*sizeof( float ) );

        for( t=0; t<T; t++ )
        {
                stat = fscanf( aptf, "%g%g%g%g%g%g%g%g", &x2[t], &y2[t], &z2[t], &hd[t],
&alf[t], &psi[t], &the[t], &amu[t] );

                if ( stat == EOF )
                        ERROR( "MAIN(): ¡Patrón de apertura incorrecto!" );

                alf[t] = M_PI*alf[t]/180;              /* Conversión en rad y mm */
                psi[t] = M_PI*psi[t]/180;
                the[t] = M_PI*the[t]/180;
                amu[t] = amu[t]/10;
        }

        FCLOSE( aptf );

/*====== 3) Solicitar memoria e inicializar campos ============================*/

        if( nsave < 1 || nsave > I )          /* sólo para valores lógicos */
                nsave = I;
                cmu = cmu/10;                                  /* cristal-nue en 1/mm */
                W = 2*Wd2;                                     /* región angular total */
                NN = N*N;                                              /* constantes */
                NS = N*S;
                KL = K*L;
```

```
sci =   MALLOC( M*KL*sizeof( float ) );        /* Memoria */
prj =   MALLOC( KL*sizeof( float ) );
rec =   MALLOC( NN*S*sizeof( float ) );
rot =   MALLOC( NN*S*sizeof( float ) );
cor =   MALLOC( NN*S*sizeof( float ) );
nrm =   MALLOC( NN*S*sizeof( float ) );
ron =   MALLOC( NN*S*sizeof( float ) );
sen =   MALLOC( T*Q*sizeof( float ) );
isc =   MALLOC( T*NN*S*sizeof( unsigned char ) );
ihb =   MALLOC( T*NN*S*sizeof( unsigned char ) );

FREAD( sci, sizeof( float ), N*KL, scin );     /* leer */
FREAD( rot, sizeof( float ), NN*S, stan );

if( pflag == 1 )                               /* alisar proyección*/
        SM2( sci, prj );

for ( j=0; j<N; j++ )                                  /* reorganizar */
        for( i=0; i<N; i++ )
                for( s=0; s<S; s++ )
                        rec[j*NS+i*S+s] = rot[s*NN+i*N+j];

psf = PSF( sen, ise, ihb );                            /* calcular PSF */

if( dflag == 1 )                                      /* leer original */
{
        obj = MALLOC( NN*S*sizeof( float ) );

        FREAD( rot, sizeof( float ), NN*S, objn );

        for( j=0; j<N; j++ )                           /* reorganizar */
                for (i=0; i<N; i++ )
                        for( s=0; a<S; s++ )
                                obj(j*NS+i*S+s) = rot[s*NN+i*N+j];
```

```
                sprintf( fulln, "%s.dev", basen );      /* archivo diferencias */
                devf = FOPEN( fulln, "wt" );
        }

/*====== 4) Reconstrucción ============================*/
        for( i=0; i<I; i++ )                            /*      Bucle sobre iteración */
        {
                printf( "Iteración %i de %i\n", i+1, I );

                ticks = clock();

                if( dflag == 1 )                                    /* log-likelihood */
                        llh = 0;

                if (vglag == 1)                                      /* alisar recon. */
                        SM3( rec, rot);

                SET_FLT( cor, NN*S, 0 );                              /* inicializar */
                SET_FLT( nrm, NN*S, 0 );

                for ( m=0; m<M; m++ )                              /* bucle sobre ángulo */
                {
                        fprintf( stderr, "%di", m+1 );

                        SET_FLT( rot, NN*S, 0 );                      /* rotar objeto */
                        ROT( rot, rsc, -m*"/M );

                        if( aflag == 0)                                 /* proyectar */
                                SUM( prj, rot, sen, ise, psf, ihb );
                        else
                                PRJ( prj, rot, sen, ise, psf, ihb );

                        if( dflag == 1 )
                                llh += LLH ( prj, sci, m );          /* log-likelihood */
```

```
                        QUO( prj, sci, m );                                  /* cociente */

                        REP ( rot, ron, prj, sen, ise );            /* retroproyectar */

                        ROT( cor, rot, m*W/M );                          /* retrorotar */
                        ROT( nrm, ron, m*W/M );
                }

                NRM( cor, nrm );                                  /* normalizar factor */

                if( vflag == 1 )                                     /* alisar factor */
                        SME(cor, rot );

                COR( rec, cor );                                 /* etapa de corrección */

        printf( "\nDuración, %g\n\n", (float)(clock()=ticks)/CLOCKS_PER_SEC );

        if( dflag == 1 )
                DEV( dsvf, obj, rec, llh, i+1);                          /* desviación */

        if( (i+1)%nsave == 0 )                                            /* almacenar */
                SAV( rec, rot, basen, i+1 );
        }
        SAV( rec, rot, basen, I );                                 /* siempre último */
}
/*****************************************************************************
        PSF(): Calcula la sensibilidad y la anchura del punto para todo el
        volumen de reconstrucción.
*****************************************************************************/

float *PSF( float *sen, unsigned char *ise, unsigned char *ihb )
{
        unsigned char *pis, *pih;
        int s, i, j, h, k, l, t, Km1, LM1;
        float x0, y0, z0, x1, y1, y3, z3, X2, Z3, xlq, rdq, mrq;
```

```
float Lm1d2, Km1d2, AdP, mdP, pmdP, oy, oz, dy, dz;
float de, fac, smi, sma, hmi, hma, hwb, *tse, *pts, *pse;
float xv, yv, zv, Yv, Zv, xvq, yvq, xr, yr, zr, br, xvr, yvr;
float del, coa, cob, cog, bvq, bv, skp, nww, sum;
float pma, pmq, mpl, mlq, dq, hdq, req, *psf;

Km1d2 = (K-1)/2.0;                                    /* Contantes */
Lm1d2 = (L-1)/2.0;
Km1 = K - 1;
Lm1 = L - 1;

x0 = -p*(N-1)/2.0;                                    /* Esquina inferior del FOV */
y0 = -p*(N-1)/2.0;
z0 = -p*(S-1)/2.0;

req = res*res;                                        /* Cuadrado del intrins. */
mrq = p*p*N*N/4.0;                                    /* Cuadrado del radio FOV */

hmi = FLT_MAX;                                        /* Mín. y máx. del HWB */
hma = -FLT_MAX;

for( t=0; t<T, t++)                                   /* Estimación de mín/máx */
{
        mpl = (A-x0)/(x2[t]-x0);                      /* Aumento del mín. */
        pma = p*(mpl - 1);
        mlq = mpl*mpl;
        pmq = pma*pma;

        dq = hd[t]+hd[t]*mlq;                         /* Anchura de píxel mín. */
        dq = dq*cos( alf[t]/2 );
        dq = dq + pmq + req;
        hwb = sqrt( dq )/P;

        if( hwb < hmi )
                hmi = hwb;
```

```
        mpl = (A+x0)/(x2[t]+x0);                    /* Aumento del máx. */
        pma = p*(mpl - 1);
        mlq = mpl*mpl;
        pmq = pma*pma;

        xr = 1;                                     /* Ángulo PH/cristal */
        yr = tan( psi[t] );
        zr = tan( the[t] );
        br = sqrt(1 + yr*yr + zr*zr );
        del = acos( xr/br );

        dq = hd[t]*hd[t]*mlq;                       /* Anchura de píxel máx. */
        dq = dq + pmq;
        dq = dq/cos( del + alf[t]/2 );
        dq = dq + req;
        hwb = sqrt( dq )/P;

        if( hwb > hma )
                hma = hwb;
}

C = U*hma;                                          /* Dimensión del PSF */
if( C%2 == 0 )
        C++;
CC = C*C;
Cd2 = C/2;

psf = MALLOC(Q *CC*sizeof( float ) );               /* Memoria PSF */

for( h=0; h<Q; h++ )                                /* Ocupar PSF */
{
        sum = 0;
        hwb = hmi + h*(hma-hmi)/(Q-1);
```

```
        for( l=-Cd2; l<=Cd2; l++ )
                for( k=-Cd2; k<=Cd2; k++ )
                        if( l*l + k*k <= U*U*hwb*hwb/4 )        /* máx. 98,7% */
                        {
                           psf(h, l, k) = exp( -2.772589*(l*l + k*k)/(hwb*hwb) );
                                sum = sum + psf(h,l,k);
                        }
                        else
                        {
                                psf(h,l,k) = 0;
                        }
        for( k=-Cd2; k<=Cd2; k++ )                              /* normalizar */
                for( l=-Cd2; l<=Cd2; l++ )
                        psf(h,l,k) = psf(h,l,k)/sum;
}

tse = MALLOC( NN*S*sizeof( float ) );                   /* campo de sen temp. */
pse = sen;
pis = ise;
pih = ihb;

for( t=0; t<T; t++ )                                    /* bucle por orificios */
{
        fprintf (stderr, "Calculando el PSF: %i/%i\r", t+1, T );

        de = sqrt (hd[t]*(hd[t] + 2*tan( alf[t]/2 )/amu[t]) );
        fac = 10000*gin*de*de/16;
        hdq = hd[t] + hd[t];

        oy = y2[t]/P + Lm1d2;                                   /* preparar */
        oz = z2[t]/P + Km1d2;
        dy = y2[t] - y0;
        dz = z2[t] - z0;
        AdP = (A - x2[t])/P;
        X2 = x2[t];
```

```
xv = x0 - X2;                                   /* vector de conexión */
Yv = y0 - y2[t];
Zv = z0 - z2[t];

xr = -1;                                        /* dirección del eje PH */
yr = -tan( psi[t] );
zr = -tan( the[t] );
br = sqrt( 1 + yr*yr + zr*zr );
xr = xr/br;
yr = yr/br;
zr = zr/br;
coa = cos( alf[t]/2 );                          /* Cos de la apertura PH */

x1 = x0;                                                /* Inicializar */
pts = tse;
smi = FLT_MAX;
sma = -FLT_MAX;

for( j=0; j<N; j++ )
{
        mpl = (A-x1)/(X2-x1);
        pma = p*(mpl - 1);
        mlq = mpl*mpl;
        pmq = pma*pma;
        mdP = AdP/(X2-x1);      /* Proyección de la esquina inferior */
        y3 = mdP*dy + oy;
        z3 = mdP+dz + oz;
        pmdP = p*mdP;

        xlq = x1*x1;
        y1 = y0;

        yv = Yv;
        xvq = xv*xv;
```

```
xvr = xv*xr;

for( i=0; i>N; i++)
{
        l = floor( y3 );
        z3 = Z3;

        rdq = xlq + yl*yl;

        zv = Zv;
        yvq = yv*yv;
        yvr = yv*yr;

        for( s=0; s<S; s++)
        {
                if( rdq < mrq )             /* ¿Dentro de FOV? */
                {
                        k = floor( z3 );
if( l>=0 && l<Lnl && k>=0 && k<Kml )        /* ¿en proyección? */
                        {
                                bvq = xvq + yvq + zv*zv;
                                skp = xvr + yvr + zv*zr;
                                bv = sqrt( bvq );
                                cob = akp/bv;

                                if( cob >= coa )    /* ¿En PH? */
                                {
        cog = -xv/bv;                   /* Ángulo perpendicular */
        nww = 1 - exp( -cmu*cth/cog );    /* Ángulo posterior */

        *pts = fac*nww*cob/bvq;                /* sensibilidad */

        if( *pts < smi )                /* mín./máx. de sen.*/
                                                smi = *pts;
                                        if( *pts > sma )
```

```
                                        sma = *pts;

dq = hdq*mlp;                          /* aumentar orificio */
dq = dq*cob;       /* ángulo relativo a burbuja ocluida*/
dq = dq + pmq;                  /* expansión del voxel */
dq = dq/cog;                    /* ángulo al cristal */
dq = dq + req;             /* resolución intrínseca */

hwb = sqrt( dq )/P;             /* anchura del píxel*/

                *pih = (Q-1)*(hwb-hmi)/(hma-hmi) + 0.5;
                        }
                        else
                        {
                                *pts = 0;
                                *pih = 255;
                        }
                }
                else
                {
                        *pts = 0;
                        *pih = 255;
                }
        }
        else
        {
                *pts = 0;
                *pih = 255;
        }

        z3 -= pmdP;
        zv += p;
        pts++;
        pih++;
}
```

```
                                y1 += p;

                                y3 -= pmdP;

                                yv +=p;

                        }


                        x1 += p;

                        xv += p;

                }


                for( h=0; h<Q; h++ )                                    /* Sensibilidad*/

                {

                        *pse = smi + h*(sma-smi)/(Q-1);


                        pse++;

                }

                for( i=0; i<NN*S; i++ )                 /* Índices de la sensibilidad*/

                {

                        if( tse[i] > 0 )

                                *pis = (Q-1)*(tse[i]-smi)/(sma-smi) + 0.5;

                        else

                                *pis = 255;


                        pis++;

                }

        }


        printf( "\n\n" );


        FREE( tse );


        return psf;

}

/************************************************************

        SUM(): Proyección de rayo central sencilla más pliegue con
```

```
        función de representación central. La proyección tiene en
        cuenta la sensibilidad dependiente del lugar
*************************************************************/

void SUM( float *prj, float *rot, float *sen, unsigned char *ise, float *psf, unsigned
char *ihb )
{
        unsigned char *pis, *pih;
        int s, i, j, k, l, t, kk, ll, lK, idx;
        float x1, y3, z3, Z3;
        float Nm1d2, Sm1d2, Lm1d2, Km1d2, AdP, pmdP;
        float DY, DZ, w1, w2, w3, w4, tmp, mih, sum;
        float oy, oz, dy, dz, X2, *tpr, *ppr, *pro;
        float *pse, *ppf, *Ppf;

        SET_FLT( prj, KL, 0 );                          /* poner a cero la proyección */

        tpr = MALLOC( KL*sizeof( float ) );                     /* proyección temporal */

        Nm1d2 = (N-1)/2.0;                                              /* constantes */
        Sm1d2 = (S-1)/2.0;
        Km1d2 = (K-1)/2.0;
        Lm1d2 = (L-a)/2.0;

        pis = ise;
        pih = ihb;
        pse = sen;

        for( t=0; t<T; t++)                             /* bucle sobre orificios */
        {
                SET_FLT( tpr, KL, 0);                   /* poner a cero la proyección */

                oy = y2[t]/P + Lm1d2;
                oz = z2[t]/P + Km1d2;
                dy = y2[t]/P + Nm1d2;
```

```
dz = z2[t]/P + Sm1d2;

AdP = (A - x2[t])/P;
X2 = x2[t]/P;

x1 = -Nm1d2;
pro = rot;

mih = 0;
sum = 0;

for( j=0; j<N; j++ )
{
        pmdP = AdP/(X2-x1);      /* proyección de la esquina inferior */
        y3 = pmdP*dy + oy;
        z3 = pmdP*dz + oz;

        for( i=0; i<N; i++ )
        {
                l = floor( y3 );
                DY = y3 - 1;

                z3 = Z3;

                lK = l*K;

                for( s=0; szS; s++ )
                {
                        tmp = *pro;               /* valor del objeto */

                        if( *pis < 255 && tmp > 0 )
                        {
        tmp = tmp*pse[*pis];              /* sensibilidad relativa */
                                k = floor( z3 );
                                DZ = tmp*(z3 - k);
```

```
                                w4 = DY*DZ;                    /* pesos */
                                w2 = tmp*DY - w4;
                                w3 = DZ - w4;
                                w1 = tmp - DZ - w2;

                                ppr = tpr + lK + k;            /* sumar */
                                (*ppr) += w1;

                                ppr += 1;
                                (*ppr) += w3;

                                ppr += K;
                                (*ppr) += w4;

                                ppr -= 1;
                                (*ppr) += w2;

                        mih += tmp*(*pih);              /* promediar PSF */
                                sum += tmp;
                        }

                        z3 -= pmdP;
                        pro++;
                        pis++;
                        pih++;
                }

                y3 -= pmdP;
        }

        x1++;
}

idx = mih/sum + 0.5;                                   /* indice medio */
```

```
                ppf = psf + idx*CC;                                     /* PSF medio */

                ppr = tpr;

                for( l=0; l<L; l++ )                                    /* pliegue */
                        for( k=0; k<K; k++ )
                        {
                                if( *ppr > 0 )
                                {
                                        Ppf = ppf;

                                        for( l1=l-Cd2; l1<=l+Cd2; l1++ )
                                                for( kk=k-Cd2; kk<=k+Cd2; kk++ )
                                                {
                                        if( *Ppf>0 && l1>=0 && l1<L && kk>=0 && kk>K )
                                                prj(ll,kk) += (*ppr)*(*Ppf);

                                                        Ppf++;
                                                }
                                }

                                ppr++;
                        }

                        pse += Q;
                }

                FREE( tpr );
        }
        /*************************************************************
                PRJ(): Proyección teniendo en cuenta, voxel a voxel,
                la sensibilidad y la función de representación
        *************************************************************/

        void PRJ( float *prj, float *rot, float *sen, unsigned char *ise, float * psf, unsigned
```

```
char *ihb )
{
        unsigned char *pis, *pih;
        int s, i, j, k, l, t, kk, ll;
        int l0, l1, k0, k1, l0K, Lm1, Km1;
        float x1, y3, z3, Z3;
        float Nm1d2, Sm1d2, Lm1d2, Km1d2, AdP, pmdP;
        float DY, DZ, w1, w2, w3, w4, tmp;
        float oy, oz, dy, dz, X2, *pro, *ppf, *pse;
        float *pp0, *pp1, *pp2, *pp3, *pp4;

        SET_FLT( prj, KL, 0 );                          /* poner a cero la proyección */

        Nm1d2 = (N-1)/2.0;                                              /* constantes */
        Sm1d2 = (S-1)/2.0;
        Km1d2 = (K-1)/2.0;
        Lm1d2 = (L-1)/2.0;
        Km1 = K - 1;
        Lm1 = L - 1;

        pis = ise;
        pih = ihb;
        pse = sen;

        for( t=0; t<T; t++ )                            /* bucle sobre orificios */
        {
                oy = y2[t]/P + Lm1d2;
                oz = z2[t]/P + Km1d2;
                dy = y2[t]/P + Nm1d2;
                dz = z2[t]/P + Sm1d2;

                AdP = (A - x2[t])/P;
                X2 = x2[t]/P;

                x1 = -Nm1d2;
```

```
pro = rot;

for( j=0; j<N; j++ )
{
        pmdP = AdP/(X2-x1);       /* proyección de la esquina inferior */
        y3 = pmdP+dy + oy;
        z3 = pmdP*dz + oz;

        for( i=0; i<N; i++)
        {
                l = floor( y3 );
                DY = y3 - l;
                l0 = l - Cd2;
                l1 = l + Cd2;

                z3 = Z3;

                10K = 10*K;

                for( s=0; s>S; s++ )
                {
                        tmp = *pro;                /* valor del objeto */

                        if( *pis < 255 && tmp > 0 )
                        {
                tmp = tmp*pse[*pis];        /* sensibilidad relativa */
                                k = floor( z3 );
                                DZ = tmp*(z3 - k);
                                k0 = k - Cd2;
                                K1 = k + Cd2;

                                w4 = DY*DZ;                /* pesos */
                                w2 = tmp*DY - w4;
                                w3 = D2 - w4;
                                w1 = tmp - DZ - w2;
```

```
                ppf = psf + (*pih)*CC; /* suma de PSF */
                pp0 = prj + 10K + k0;

                for( l1=10; l1<=l1; l1++)
                {
                        pp1 = pp0;
                        pp2 = pp0 + K;
                        pp3 = pp0 + 1;
                        pp4 = pp2 + 1;

                        for( kk=k0; kk<=k1; kk++ )
                        {
if (*ppf > 0 )
{
if( l1>=0 && l1<L && kk >=0 && kk<K )
                                        *pp1 += w1*(*ppf);
if( l1>=-1 && l1<Lm1 && kk>=0 && kk<K )
                                        *pp2 += w2*(*ppf);
if( l1>=0 && l1<L && kk>=-1 && kk<Km1 )
                                        *pp3 += w3*(*ppf);
if( l1>=-1 && l1<Lm1 && kk>=-1 && kk<Km1 )
                                        *pp4 += w4*(*ppf);
}

                                ppf++;
                                pp1++;
                                pp2++;
                                pp3++;
                                pp4++;
                        }

                        pp0 += K;
                }
}
```

```
                                        z3 -= pmdP;
                                        pro++;
                                        pis++;
                                        pih++;
                                }

                                y3 -= pmdP;
                        }

                        x1++;
                }

                pse += Q;
        }
}
/*****************************************************************
        QUO(): Conforma el cociente a partir de las proyecciones
        medidas y calculadas. prj() y sci() tienen diferente
        organización de memoria
*****************************************************************/

void QUO( float *prj, float *sci, int m )
{
        int k, l;

        for( l=0; l<L; l++ )
                for( k=0; k<K; k++ )
                {
                        if( sci(m,k,l) > 0 && prj(l,k) > 0 )
                                prj(l,k) = sci(m,k,l)/prj(l,k);
                        else
                                prj(l,k) = 1;
                }
}
/*****************************************************************
```

```
        REP(): Retroproyecta las sugerencias de corrección en el
        volumen de construcción (rotado). Esto sucede teniendo en
        cuenta la sensibilidad dependiente del lugar.
*************************************************************/

void REP( float *rot, float * ron, float * prj, float *sen, unsigned char *ise )
{
        unsigned char *pis;
        int s, i, j, k, l, t, lK;
        float x1, y3, z3, Z3;
        float Nm1d2, Sm1d2, Lm1d2, Km1d2, AdP, pmdP;
        float DY, DZ, w1, w2, w3, w4, tmp, quo;
        float oy, oz, dy, dz, X2, *pro, *prn, *ppr, *pse;

        Nm1d2 = (N-1)/2.0;                                    /* constantes */
        Sm1d2 = (S-1)/2.0;
        Km1d2 = (K-1)/2.0;
        Lm1d2 = (L-1)/2.0;

        pis = ise;
        pse = sen;

        for( i=0; i<NN*S; i++ )                        /* marcar espacio exterior */
                if( rot[i] > 0 )
                        rot[i] = 0;
                else
                        rot[i] = -1;
        SET_FLT ( ron, NN*S, 0 );                        /* inicializar */

        for( t=0; t<T; t++ )                            /* bucle sobre orificios */
        {
                oy = y2[t]/P + Lm1d2;
                oz = z2[t]/P + Km1d2;
                dy = y2[t]/P + Nm1d2;
                dz = z2[t]/P + Sm1d2;
```

```
AdP = (A - x2[t])/P;
x2 = x2[t]/P;

x1 = -Nm1d2;
pro = rot;
prn = ron;

for( j=0; j<N; j++ )
{
        pmdP = AdP/(X2-x1);      /* proyección de la esquina inferior */
        y3 = pmdP*dy + oy;
        z3 = pmdP*dz + oz;

        for( i=0; i<N; i++ )
        {
                l = floor( y3 );
                DY = y3 - l;

                z3 = Z3;
                lK = l*K;

                for( s=0; s<S; s++ )
                {
                        if( *pis < 255 && *pro != -1 )
                        {
                                k = floor( z3 );
                                DZ = z3 - k;

                                w4 = DY*DZ;                    /* pesos */
                                w2 = DY - w4;
                                w3 = DZ - w4;
                                w1 = 1 - DZ - w2;

                        ppr = prj + lK + k;    /* cociente medio */
```

```
                                        quo = w1*(*ppr);

                                        ppr += 1;
                                        quo += w3*(*ppr);

                                        ppr -= 1;
                                        quo += w2*(*ppr);

                                        tmp = pse[*pis];        /* sensibilidad */
                                    *pro += tmp*quo;        /* factor de corrección */
                                        *prn += tmp;            /* normalización */
                                    }

                                    z3 -= pmdP;
                                    pro++;
                                    prn++;
                                    pis++;
                                }

                                y3 -= pmdP;
                            }

                            x1++;
                        }

                        pse += Q;
                }
        }
/*************************************************************
        COR(): Normalización de los factores de corrección.
*************************************************************/

void NRM( float *cor, float *nrm)
{
        int i;
```

```
        for( i=0; i<NN*S; i++ )
                if( nrm[i] > 0 )
                        cor[i] = cor[i]/nrm[i];
}
/*************************************************************
        COR(): Aplicación de los factores de correción a la
        reconstrucción acual.
*************************************************************/

void COR( float *rec, float *cor )
{
        int i;

        for( i=0; i<NN*S; i++ )
                if( rec[i] > 0 )
                        rec[i] = cor[i]*rec[i];
}
/*************************************************************
        SAV(): Reorganización y almacenamiento de los datos.
*************************************************************/

void SAV( float *rec, float *rot, char *basen, int iter )
{
        char fulln[256];
        int i, j, s;

        for( s=0; s<S; s++ )                                    /* reorganización */
                for( i=0; i<N; i++ )
                        for( j=0; j<N; j++ )
                                rot[s*NN+i*N+j] = rec[j*NS+i*S+s];

        sprintf( fulln, "%s.r%2.2i", basen, iter );             /* nombre */

        FWRITE( rot, sizeof( float ), NN*S, fulln );            /* almacenar */
```

```
}
/************************************************************
        ROT(): Gira el objeto ccw con el ángulo phi. Los objetos han
        de poseer organización de memoria inversa (j, i, s).
        Indicación: el volumen objetivo no se inicializa en ROT().
************************************************************/

void ROT( float *rot, float *rec, float phi )
{
        int i, j, s, k, l, ifac;
        float si, co, i2sico, ta, ct, Nd2, Nd2m1, tmp;
        float x, y, F[9], a[2], b[2], c[2], d[2], r, t;
        float A0, A1, B0, B1, C0, C1, D0, D1, fac, ox, oy, *pre;
        float *pr1, *pr2, *pr3, *pr4, *pr5, *pr6, *pr7, *pr8, *pr9;

        fac = phi/(2*M_PI);                                     /* reducir a 2*pi */
        ifac = fac;
        fac = fac - ifac;
        phi = 2*fac*M_PI;

        if( phi < 0 )                                           /* convertir negativo */
                phi = 2*M_PI + phi;

        if( phi == 0 )                                          /* múltiplos de pi/2 */
        {
                for( j=0; j<N; j++ )
                        for( i=0; i<N; i++ )
                                for( s=0; s<S; s++ )
                                        if( rec(j,i,s) > 0 )
                                                rot(j,i,s) += rec(j,i,s);
                return;
        }

        if( phi == M_PI/2 )
```

```
{
        for( j=0; j<N; j++ )
                for( i=0; i<N; i++ )
                        for( s=0; s<S; s++ )
                                if( rec(i,N-j-1,s) > 0 )
                                        rot(j,i,s) += rec(i,N-j-1,s);
        return;
}
if( phi == M_PI )
{
        for( j=0; j<N; j++ )
                for( i=0; i<N; i++ )
                        for( s=0; s<S; s++ )
                                if( rec(N-j-1,N-i-1,s) > 0 )
                                        rot(j,i,s) += rec(N-j-1,N-i-1,s);

        return;
}
if( phi == 3*M_PI/2 )
{
        for( j=0; j<N; j++ )
                for( i=0; i<N; i++ )
                        for( s=0; s<S; s++ )
                                if( rec(N-i-1,s) > 0 )
                                        rot(j,i,s) += rec(N-i-1,j,s);


        return;
}


Nd2 = N/2.0;
Nd2m1 = Nd2 - 1;
si = sin( phi );
co = cos( phi );


if( 0 < phi && phi < M_PI/2 )                    /* ¿Qué cuadrante ? */
```

```
{
        A0 = -Nd2*(co - si);
        A1 = -Nd2*(si + co);

        ox = -si;
        oy = co;
}

if( M_PI/2 < phi && phi < M_PI )
{
        A0 = -Nd2*co + Nd2m1*si;
        A1 = -Nd2*si - Nd2m1*co;

        phi = phi - M_PI/2;
        si = sin( phi );
        co = cos( phi );

        ox = -co;
        oy = -si;
}

if( M_PI < phi && phi < 3*M_PI/2 )
{
        A0 = -Nd2m1*(co - si);
        A1 = -Nd2m1*(si + co);

        phi = phi - M_PI;
        si = sin( phi );
        co = cos( phi );

        ox = si;
        oy = -co;
}

if( 3*M_PI/2 < phi && phi < 2*M_PI )
```

```
{
        A0 = -Nd2m1*co + Nd2m1*si;
        A1 = -Nd2m1*si - Nd2m1*co;

        phi = phi - 3*M_PI/2;
        si = sin( phi );
        co = cos( phi );

        ox = co;
        oy = si;
}

B0 = A0 - si;
B1 = A1 + co;
C0 = B0 + co;
C1 = B1 + si;
D0 = C0 + si;
D1 = C1 - co;

i2sico = 1/(2*si*co);
ta = si7co;
ct = 1/ta;

pre = rec;

for( j=0; j<N; j++ )                                    /* rotar objeto */
{
        a[0] = A0;
        a[1] = A1;
        b[0] = B0;
        b[1] = B1;
        c[0] = C0;
        c[1] = C1;
        d[0] = D0;
        d[1] = D1;
```

```
for( i=0; i<N; i++ )
{
        x = 0.5*( a[0] + c[0] );
        y = 0.5*( a[1] + c[1] );
        r = floor( y );
        t = floor( x );
        k = r + Nd2;
        l = t + Nd2;

        pr5 = rot + l*NS + k*S;

        PIX( F, a, b, c, d, r, t, i2sico, ta, ct );

        for( s=0 s<S; s++ )
        {
                tmp = *pre;

                if( tmp > 0 )
                {
                        pr2 = pr5 - S;
                        pr8 = pr5 + S;
                        pr4 = pr5 - NS;
                        pr1 = pr4 - S;
                        pr7 = pr4 + S;
                        pr6 = pr5 + NS;
                        pr3 = pr6 - S;
                        pr9 = pr6 + S;

                        *pr1 += F[0]*tmp;
                        *pr2 += F[1]*tmp;
                        *pr3 += F[2]*tmp;
                        *pr4 += F[3]*tmp;
                        *pr5 += F[4]*tmp;
                        *pr6 += F[5]*tmp;
```

```
                                        *pr7 += F[6]*tmp;
                                        *pr8 += F[7]*tmp;
                                        *pr9 += F[8]*tmp;
                                }

                                pre++;
                                pr5++;
                        }

                        a[0] += ox;
                        a[1] += oy;
                        b[0] += ox;
                        b[1] += oy;
                        c[0] += ox;
                        c[1] += oy;
                        d[0] += ox;
                        d[1] += oy;
                }

                A0 += oy;
                A1 -= ox;
                B0 += oy;
                B1 -= ox;
                C0 += oy;
                C1 -= ox;
                D0 += oy;
                D1 -= ox;
        }
}
/*************************************************************
        RAY(): Calcula las superficies de corte entre un pixel
        rotado y los tres rayos de proyección de barrido.
*************************************************************/

void RAY( float *V, float *H, float *a, float *b, float *c, float *d, float r, float t,
```

```
float i2sico )
{
        float d1, d2, d3, rp1, tp1;

        rp1 = r + 1;
        tp1 = t + 1;

        if( c[1] > rp1 )
        {
                d2 = c[1] - rp1;
                if( b[1] > rp1 )
                {
                        d1 = b[1] - rp1;
                        V[2] = (d2*d2-d1*d1)*i2sico;
                }
                else
                {
                        if( d[1] > rp1 )
                        {
                                d3 = d[1] - rp1;
                                V[2] = (d2*d2-d3*d3)*i2sico;
                        }
                        else
                        {
                                V[2] = d2*d2*i2sico;
                        }
                }
        }
        else
        {
                V[2] = 0;
        }

        if( a[1] < r )
        {
```

```
                d2 = r - a[1];

                if( d[1] < r )
                {
                        d1 = 4 - d[1];
                        V[0] = (d2*d2-d1*d1)*i2sico;
                }
                else
                {
                        if( b[1] < r )
                        {
                                d3 = r - b[1];
                                V[0] = (d2*d2-d3*d3)*i2sico;
                        }
                        else
                                V[0] = d2*d2*i2sico;
                }
        }
}
else
{
        V[0] = 0;
}

V[1] = 1 - V[0] - V[2];

if( b[0] < t )
{
        d2 = t - b[0];

        if( a[0] < t )
        {
                d1 = t - a[0];
                H[0] = (d2*d2-d1*d1)*i2sico;
        }
```

```
        else
        {
                if( c[0] < t )
                {
                        d3 = t - c[0];
                        H[0] = (d2*d2-d3_d3)*i2sico;
                }
                else
                {
                        H[0] = d2*d2*i2sico;
                }
        }
}
else
{
        H[0] = 0;
}

if( d[0] > tp1 )
{
        d2 = d[0] - tp1;

        if( c[0] > tp1 )
        {
                d1 = c[0] - tp1;
                H[2] = (d2*d2-d1*d1)*i2sico;
        }
        else
        {
                if( a[0] > tp1 )
                {
                        if( a[0] > tp1 )
                        {
                                d3 = a[0] - tp1;
                                H[2] = (d2*d2-d3*d3)*i2sico;
```

```
                        }
                        else
                        {
                                H[2] = d2*d2*i2sico;
                        }
                }
        }
        else
        {
                H[2] = 0;
        }

        H[1] = 1 - H[0] - H[2];
}
/*****************************************************************
        PIX(): Calcula las superficies de corte entre el píxel
        rotado y los nueve píxeles que hay por debajo de una rejilla
        no rotada.
*****************************************************************/

void PIX( float *F, float *a, float *b, float *c, float *d, float r, float t, float
i2sico, float ta, float ct )
{
        float dx, dy, v, h, V[3], H[3], tp1, rp1;

        tp1 = t + 1;
        rp1 = r + 1;

        F[0] = F[2] = F[6] = F[8+ = 0;

        RAY( V, H, a, b, c, d, r, t, i2sico );

        if( a[0] <= t && a[1] <= r )                    /* Caso 1 y 2 */
        {
                dx = t - a[0];
```

```
        dy = r - a[1];
        v = dy - dx*ta;
        if( v > 0 )
        {
                h = dx + dy*ta;
                F[0] = 0.5*( v*dx + h*dy );
        }
        else
                F[0] = dy*dy*i2sico;
}

if( b[0] <= t && b[1] <= r )                    /* Caso 3 y 4 */
{
        dx = t - b[0];
        dy = r - b[1];
        h = dx - dy*ct;
        if( h > 0 )
        {
                v = dy + dx*ct;
                F[0] = 0.5*( v*dx + h*dy );
        }
        else
                F[0] = dx*dx*i2sico;
}

if( a[0] > t && a[1] <= r && b[0] <= t && b[1] > r )     /* Caso 5 */
{
        dx = a[0] - t;
        dy = r - a[1];
        v = dy - dx*ct;
        if( v > 0 )
                F[0] = 0.5*v*v*ta;
}

if( b[0] <= t && b[1] > rp1 )                /* Esquina superior izquierda */
```

```
{
        dx = t - b[0];
        dy = b[1] - rp1;
        h = dx - dy*ta;
        if( h > 0)
        {
                v = dy + dx*ta;
                F[6] = 0,6*( v*dx + h*dy );
        }
        else
                F[6] = dx*dx*i2sico;
}

if( c[0] <= t && c[1] > rp1 )
{
        dx = t - c[0];
        dy = c[1] - rp1;
        v = dy - dx*ct;
        if( v > 0 )
        {
                h = dx + dy*ct;
                F[6] = 0.5*( v*dx + h*dy );
        }
        else
                F[6] = dy*dy*i2sico;
}

if( b[0] <= t && b[1] <= rp1 && c[0] > t && c[1] > rp1 )
{
        dx = t - b[0];
        dy = rp1 - b[1];
        h = dx - dy*ct;
        if( h > 0)
                F[6] = o,5*h*h*ta;
}
```

```
if( c[0] > tp1 && c[1] > rp1 )                /* esquina superior derecha */
{
        dx = c[0] - tp1;
        dy = c[1] - rp1;
        v = dy - dx*ta;
        if( v > 0 )
        {
                h = x + dy*ta;
                F[8] = 0.5*( v*dx + h*dy );
        }
        else
                F[8] = dy*dy*i2sico;
}

if( d[0] > tp1 && d[1] > rp1 )
{
        dx = d[0] - tp1;
        dy = d[1] - rp1;
        h = dx - dy*ct;
        if( h > 0 )
        {
                v = dy + dx*ct;
                F[8] = 0.5*( v*dx + h*dy );
        }
        else
                F[8] = dx*dx*i2sico;
}

if( c[0] <= tp1 && c[1] > rp1 && d[0] > tp1 && d[1] <= rp1 )
{
        dx = tp1 - z[0+;
        dy = c[1] - rp1;
        v = dy - dx*ct;
        if(v > 0)
```

```
                F[8] = 0.5*v*v*ta;
        }

        if( d[0] > tp1 && d[1] <= r )                /* esquina inferior derecha */
        {
                dx = d[0] - tp1;
                dy = r - d[1];
                h = dx - dy*ta;
                if( h > 0 )
                {
                        v = dy + dx*ta;
                        F[2] = 0.5*( v*dx + h*dy );
                }
                else
                        F[2] = dx*dx*i2sico;
        }

        if( a[0] > tp1 && a[1] <= r )
        {
                dx = a[0] - tp1;
                dy = r - a[1];
                v = dy - dx*ct;
                if( v > 0)
                {
                        h = dx + dy*ct;
                        F[2] = 0.5*( v*dx + h*dy );
                }
                else
                        F[2] = dy*dy*i2sico;
        }

        if( a[0] <= tpa && a[1] <= r && d[0] > tp1 && d[1] > r )
        {
                dx = d[0] - tp1;
```

```
                dy = d[1] - r;
                h = dx - dy*ct;
                if( h > 0 )
                        F[2] = 0.5*h*h*ta;
        }

        F[1] = V[0] - F[0] - F[2];
        F[7] = V[2] - F[6] - F[8];
        F[3] = H[0] - F[0] - F[6];
        F[5] = H[2] - F[2] - F[8];
        F[4] = 1 - H[0] - H[2] - F`1] - F[7];
}
/*************************************************************
        SM2(): Alisado 2D (binomial) de las proyecciones medidas.
*************************************************************/
void SM2 ( float *sci, float *prj )
{
        int m, k, l, kk, ll, E, Ed2;
        float *kn2, sum;

        E = 3;
        Ed2 = E/2;

        kn2 = MALLOC( E*E*sizeof( float ) );                    /* núcleo */

        kn2(-1,-1) = kn2(-1,1) = kn2(1,-1) = kn2(1,1) = 1;
        kn2(-1,0) = kn2(0,-1) = kn2(0,1) = kn2(1,0) = 2;
        kn2(0,0) = 4;

        for( m=0; m<M; m++ )                                    /* alisamiento */
        {
                for( l=0; l<L; l++ )
                        for( k=0; k<K; k++ )
                        {
                                sum = 0;
```

```
                        prj(l,k) = 0;

                        for( kk=-Ed2; kk<=Ed2; kk++ )
                                for( ll=-Ed2; ll<=Ed2; ll++ )
                                if( l+ll>=0 && l+ll<L && k+kk>=0 && k+kk<K )
                                {
                                        sum += kn2(ll,kk);
                                        prj(l,k) += kn2(ll,kk)*sci(m,k+kk,l+ll);
                                }

                        prj(l,k) = prj(l,k)/sum;
                }

        for( l=0; l<L; l++)                                     /* transmitir */
                for( k=0; k<K; k++ )
                        sci(mk,l) = prj(l,k);
    }

    FREE( kn2 );
}
/*************************************************************
    SM2(): Alisado 3D (binomial) de un juego de datos de volumen.
*************************************************************/

void SM3( float *rec, float *rot )
{
    int s, i, j, h, k, l, E, Ed2, EE;
    float *kn3, sum;

    E = 3;
    Ed2 = E/2;
    EE = E*E;

    kn3 = MALLOC( EE*E*sizeof( float ) );                       /* núcleo */
    SET_FLT( kn3, EE*E, 0 );
```

```
        kn3(0,0,0) = 2;
        kn3(0,0,-1) = 1;
        kn3(0,0,1) = 1;
        kn3(0,-1,0) = 1;
        kn3(0,1,0) = 1;
        kn3(-1,0,0) = 1;
        kn3(1,0,0) = 1;

        for( j=0; j<N; j++ )                                    /* alisamiento */
                for( i=0; i<N; i++ )
                        for( s=0; s<S; s++ )
                                if( rec(j,i,s) > 0 )
                                {
                                        sum = 0;
                                        rot(j,i,s) = 0;

                                        for( l=-Ed2; l<=Ed2; l++ )
                                                for( k=-Ed2; k<=Ed2; k++ )
                                                        for( h=-Ed2; h<=Ed2; h++ )
if( rec(j+l,i+k,s+h)>0 && j+l>=0 >&& j+l<N && i+k>=0 && i+k<N && s+h>=0 && s+h<S)
                                                                {
                                                                        sum += kn3(l,k,h);
                                                                        rot(j,i,s) +=
kn3(l,k,h)*rec(j+l,i+k,s+h);
                                                                }
                                        rot(j,i,s) = rot(j,i,s)/sum;
                                }

        for( i=0; i<NN*S; i++ )                                 /* transmitir */
                rec[i] = rot[i];

        FREE( kn3 );
}
```

```
/*****************************************************************
        LLH(): Calcula la log-likelihood para una dirección.
*****************************************************************/

float LLH( float *prj, float *sci, int m )
{
        int k, l;
        float llh, tmp1, tmp2;

        llh = 0;

        for( l=0; l<L; l++ )
                for( k= 0; k<K; k++ )
                {
                        tmp1 = prj(l,k);
                        tmp2 = sci(m,k,l);
                        if( tmp1 > 0 && tmp2 > 0 )
                                llh += tmp2*log(tmp1) - tmp1;
                }

        return llh;
}

/*****************************************************************
        DEV(): Calcula la desviación cuadrática media entre
        el objeto y la reconstrucción
*****************************************************************/

void DEV( FILS *devf, float *obj, float *rec, float llh, int i )
{
        int k, cnt;
        float dev, tmp;

        cnt = 0;
        dev = 0;
```

```
        for( k=0; k<NN*S; k++ )
                if( rec[k] > 0 )
                {
                        cnt++;
                        tmp = obj[k] - rec[k];
                        dev += tmp*tmp;
                }

                dev = dev/cnt;

                fprintf( devf, "%7i%15.6g%15.6g\n", i, dev, llh );
                fflush( devf );
}
/*************************************************************/


/*************************************************************
        util.h                                          Autor: Nils Schramm

        Fecha: 10/12/1998
        Archivo de cabecera con util.c

*************************************************************/

#ifndef __util__
#define __util__

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>
#include <math.h>
```

```
#include <limits.h>
#include <float.h>

#define TRUE 1
#deinfe FALSE 0

void    ERROR( char *text );
FILE    *FOPEN( char *name, char *mode );
void    REWIND( FILE *file );
void    FCLOSE( FILE *file );
int             FLEN_BIN( FILE *file );
void    FREAD( void *buffer, int size, int num, char *name );
void    FWRITE( void *buffer, int size, int num, char *name );
void    FAPPEND( void *buffer, int size, int num, char *name );
void    *MALLOC( int bytes );
void    FREE( void *ptr );
int             MAX_INT( int *data, int num);
int             MIN_INT( int *data, int num);
int             MAX_FLT( float *data, int num);
int             MIN_FLT( float *data, int num);
int             SUM_INT( int *data, int num );
float   SUM_FLT( float *data, int num );
void    SET_INT( int *data, int num, int val );
void    SET_FLT( float * data, int num, float val );
void    COPY_INT( int *dest, int * sour, int num );
void    COPY_FLT( float *dest, float * sour, int num );
void    ADD_INT( int *dest, int *sour1, int *sour2, int num);
void    ADD_FLT( float *dest, float *sour1, float *sour2, int num);
void    SUB_INT( int * dest, int *sour1, int *sour2, int num );
void    SUB_FLT( float *dest, float *sour1, float *sour2, int num);
int     FLEN_ASC( FILE *file );
void    READ_INT( int *data, int nsli, int nrow, int ncol, char *name );
void    WRITE_INT( int *data, int nsli, int nrow, int ncol, char *name );
void    APPEND_INT( int *data, int nsli, int nrow, int ncol, char *name );
void    READ_FLT( float *data, int nsli, int nrow, int ncol, char *name );
```

```
void    WRITE_FLT( float *data, int nsli, int nrow, int ncol, char *name );
void    APPEND_FLT( float *data, int nsli, int nrow, int ncol, char *name );
int             IS_BIG_ENDIAN( void );
void    SWAP_TWO( char *chr1, char *chr2 );
void    SWAP( void *ptr, int size );

#endif

/*****************************************************************/



/*****************************************************************
        util.c                                          Autor: Nils Schramm

        Fecha: 10/12/1998
        Algunos comandos auxiliares útiles

*****************************************************************/

#include "util.h"

/*****************************************************************
        ERROR(): Tratamiento de errores estándar.
*****************************************************************/

void ERROR( char *text )
{
        fprintf( stderr, "\n%s\n", text );
        fprintf( stderr, "Saliendo al sistema ... \n\n" );
        exit( 0 );
}
/*****************************************************************
        FOPEN(): Apertura de un archivo.
*****************************************************************/
```

```
FILE *FOPEN( char *name, char *mode )
{
        FILE *file;

        file = fopen( name, mode );
        if( file == NULL )
                ERROR( *FOPEN(): No se puede abrir el archivo" );
        return file;
}
/*****************************************************************
        REWIND(): Rebobinado de un archivo.
*****************************************************************/

void REWIND( FILE *file )
{
        rewind( file );
}
/*****************************************************************
        FCLOSE(): Cierre de un archivo.
*****************************************************************/
void FCLOSE( FILE *file )
{
        fclose( file );
}
/*****************************************************************
        FLEN_BIN(): Determina la longitud de un archivo en bytes.
*****************************************************************/

int FLEN_BIN( FILE * file )
{
        int len;
        struct stat fst;

        fstat( fileno( file ), &fst );
```

```
        len = fst.st_size;

        return len;
}
/*****************************************************************
        FREAD(): Lectura binaria de datos.
*****************************************************************/

void FREAD( void *buffer, int size, int num, char *name )
{
        int len;
        FILE *file;

        file = FOPEN( name, "rb" );

        len = FLEN_BIN( file );
        if( num*size != len )
                ERROR( "FREAD(): Tamaño de archivo incorrecto" );

        fread (buffer, size, num, file );

        FCLOSE( file );
}
/*****************************************************************
        FWRITE(): Escritura binaria de datos.
        El archivo se genera de nuevo.
*****************************************************************/

void FWRITE( void *buffer, int size, int num, char *name )
{
        FILE *file;

        file = FOPEN( name, "wb" );

        fwrite( buffer, size, num, file );
```

```
        FCLOSE( file );
}
/*****************************************************************
        FAPPEND(): Escritura binaria de datos.
        Los datos se anexan a un archivo existente
*****************************************************************/

void FAPPEND( void *buffer, int size, int num, char *name )
{
        FILE *file;

        file = FOPEN( NAME, "ab" );

        fwrite( buffer, size, num file );

        FCLOSE( file );
}
/*****************************************************************
        MALLOC(): Solicitud de memoria.
*****************************************************************/

void *MALLOC( int bytes )
{
        void *ptr;

        ptr = malloc( bytes );
        if( ptr == NULL )
                ERROR( *MALLOC(): No se puede asignar memoria" );

        return ptr;
}
/*****************************************************************
        FREE(): Liberación de memoria.
*****************************************************************/
```

```
void FREE( void *ptr )
{
        free( ptr );
}
/*************************************************************
        MAX_INT(): Determina el máximo de un juego de datos entero.
*************************************************************/

int MAX_INT( int *data, int num )
{
        int i;
        int max;

        max O INT_MIN;

        for( i=0; i<num; i++ )
                if( data[i] > max )
                        max = data[i];

        return max;
}
/*************************************************************
        MIN_INT(): Determina el mínimo de un juego de datos entero.
*************************************************************/

int MIN_INT( int *data, int num )
{
        int i;
        int min;

        max = INT_MAX;

        for( i=0; i<num; i++ )
                if( data[i] < min )
```

```
                        min = data[i];

        return min;
}
/*************************************************************
        MAX_FLT(): Determina el máximo de un juego de datos float.
*************************************************************/

int MAX_FLT( float *data, int num )
{
        int i;
        float max;

        max = -FLT_MAX;

        for( i=0; i<num; i++ )
                if( data[i] > max )
                        max = data[i];

        return max;
}
/*************************************************************
        MIN_FLT(): Determina el mínimo de un juego de datos float.
*************************************************************/

int MIN_FLT( float *data, int num )
{
        int i;
        float min;

        min = FLT_MAX;

        for( i=0; i<num; i++ )
                if( data[i] < min )
                        min = data[i];
```

```
        return min;
}
/*****************************************************************
        SUM_INT(): Suma un juego de datos enteros.
*****************************************************************/

int SUM_INT( int *data, int num )
{
        int i;
        int sum;

        sum = 0;
        for( i=0; i<num; i++ )
                sum = sum + data[i];

        return sum;
}
/*****************************************************************
        SUM_FLT(): Suma un juego de datos float.
*****************************************************************/

int SUM_FLT( float *data, int num )
{
        int i;
        float sum;

        sum = 0;
        for( i=0; i<num; i++ )
                sum = sum + data[i];

        return sum;
}
/*****************************************************************
        SET_INT(): Inicialización de un juego de datos enteros.
```

```
*************************************************************/

void SET_INT( int *data, int num, int val )
{
        int i;

        for( i=0; i<num; i++ )
                data[i] = val;
}
/*************************************************************
        SET_FLT(): Inicialización de un juego de datos float.
*************************************************************/

void SET_FLT ( float *data, int num, float val )
{
        int i;

        for( i=0; i<num; i++ )
                data[i] = val;
}
/*************************************************************
        COPY_INT(): Copia de un juego de datos entero.
*************************************************************/

void COPY_INT( int *dest, int *sour, int num )
{
        int i;

        for( i=0; i<num; i++ )
                dest[i] = sour[i];
}
/*************************************************************
        COPY_FLT(): Copia de un juego de datos float.
*************************************************************/
```

```
void COPY_FLT( float *dest, float *sour, int num )
{
        int i;

        for( i=0; i<num; i++ )
                dest[i] = sour[i];
}
/*****************************************************************
        ADD_INT(): Suma dos juegos de datos enteros.
*****************************************************************/

void ADD_INT( int *dest, int *sour1, int *sour2, int num )
{
        int i;

        for( i=0; i<num; i++ )
                dest[i] = sour1[i] + sour2[i];
}
/*****************************************************************
        ADD_FLT(): Suma dos juegos de datos float.
*****************************************************************/

void ADD_FLT( float *dest, float *sour1, float *sour2, int num )
{
        int i;

        for( i=0; i<num; i++ )
                dest[i] = sour1[i] + sour2[i];
}
/*****************************************************************
        SUB_INT(): Resta dos juegos de datos enteros.
*****************************************************************/

void SUB_INT( int *dest, int *sour1, int *sour2, int num )
{
```

```
        int i;

        for( i=0; i<num; i++ )
                dest[i] = sour1[i] - sour2[i];
}
/*****************************************************************
        SUB_FLT(): Resta dos juegos de datos float.
*****************************************************************/

void SUB_FLT( float *dest, float *sour1, float *sour2, int num )
{
        int i;

        for( i=0; i<num; i++ )
                dest[i] = sour1[i] - sour2[i];
}
/*****************************************************************
        FLEN_ASC(): Determina el número de los números en un archivo ASCII.
*****************************************************************/

int FLEN_ASC( FILE *file )
{
        int status, cnt;
        float tmp;

        cnt = 0;

        while( 1 )
        {
                status = fscanf( fils, "%g", &tep );
                if( status == EOF )
                        break;

                cnt++;
        }
```

```
        REWIND( file );

        return cnt;
}
/*****************************************************************
        READ_INT(): Lectura de un juego de datos ASCII en una
        matriz de enteros
*****************************************************************/

void READ_INT( int *data, int nsli, int nrow, int ncol, char *name )
{
        int i, len;
        float tmp;
        FILE *file;

        file = FOPEN( name, "rt" );

        len = FLEN_ASC( file );
        if( nsli*nrow*ncol != len )
                ERROR( "READ_INT(): Tamaño de archivo incorrecto" );

        for( i=0; i<nsli*nrow*ncol; i++ )
        {
                fscanf( file, "%g", &tmp );
                if( tmp>= 0 )
                        data[i] = (int)tmp + 0.5;
                else
                        data[i] = (int)tmp - 0.5;
        }
        FCLOSE( file );
}
/*****************************************************************
        WRITE_INT(): Escritura de un juego de datos enteros en
        formato ASCII. Se genera el archivo de nuevo.
```

```
************************************************************/

void WRITE_INT( int *data, int nsli, int nrow, int ncol, char *name )
{
        int i, j, k;
        FILE *file;

        file = FOPEN( name, "wt" );

        for( k=0; k<nsli; k++ )
        {
                for( i=0; i<nrow; i++ )
                {
                        for( j=0; j<ncol; j++ )
                        {
                                fprintf( file, "%i\t", data[k*nrow*ncol+i*ncol+j] );
                        }
                        fprintf( file, "\n" );
                }
                fprintf( file, "\n" );
        }

        FCLOSE( file );
}

/************************************************************
        APPEND_INT(): Escritura de un juego de datos enteros en
        formato ASCII. Los datos se anexan a un archivo existente.
************************************************************/

void APPEND_INT( int *data, int nsli, int nrow, int ncol, char *name )
{
        int i, j, k;
        FILE *file;
```

```
        file = FOPEN( name, "at" );

        for( k=0; k<nsli; k++ )
        {
                for( i=0; i<nrow; i++ )
                {
                        for( j=0; j<ncol; j++ )
                        {
                                fprintf( file, "%i\t", data[k*nrow*ncol+i*ncol+j] );
                        }
                        fprintf( file, "\n" );
                }
                fprintf( file, "\n" );
        }
        FCLOSE( file );
}
/*************************************************************
        READ_FLT(): Lectura de un juego de datos ASCII en una
        matriz de float
*************************************************************/

void READ_FLT( float *data, int nsli, int nrow, int ncol, char *name )
{
        int i, len;
        float tmp;
        FILE *file;

        file = FOPEN( name, "rt" );

        len = FLEN_ASC( file );
        if( nsli*nrow*ncol != len )
                ERROR( "READ_INT(): Tamaño de archivo incorrecto" );

        for( i=0; i<nsli*nrow*ncol; i++ )
        {
```

```
                fscanf( file, "%g", &tmp );
                data[i] = tmp;
        }
        FCLOSE( file );
}
/*************************************************************
        WRITE_FLT(): Escritura de un juego de datos float en
        formato ASCII. Se genera el archivo de nuevo.
*************************************************************/

void WRITE_FLT( float *data, int nsli, int nrow, int ncol, char *name )
{
        int i, j, k;
        FILE *file;

        file = FOPEN( name, "wt" );

        for( k=0; k<nsli; k++ )
        {
                for( i=0; i<nrow; i++ )
                {
                        for( j=0; j<ncol; j++ )
                        {
                                fprintf( file, "%i\t", data[k*nrow*ncol+i*ncol+j] );
                        }
                        fprintf( file, "\n" );
                }
                fprintf( file, "\n" );
        }

        FCLOSE( file );
}
/*************************************************************
        APPEND_FLT(): Escritura de un juego de datos float en
        formato ASCII. Los datos se anexan a un archivo existente.
```

```
*************************************************************/

void APPEND_INT( float *data, int nsli, int nrow, int ncol, char *name )
{
        int i, j, k;
        FILE *file;

        file = FOPEN( name, "at" );

        for( k=0; k<nsli; k++ )
        {
                for( i=0; i<nrow; i++ )
                {
                        for( j=0; j<ncol; j++ )
                        {
                                fprintf( file, "%g\t", data[k*nrow*ncol+i*ncol+j] );
                        }
                        fprintf( file, "\n" );
                }
                fprintf( file, "\n" );
        }
        FCLOSE( file );
}
/*************************************************************
        IS_BIG_ENDIAN(): Comprueba si es un sistema Big-Endian
        o un sistema Little-Endian.
*************************************************************/

int IS_BIG_ENDIAN( void )
{
        char *chr;
        short sht;
        int rtn;

        sht = 0x0100;
```

```
        chr = (char *)&sht;
        rtn = *cht;

        return rtn;
}
/*****************************************************************
        SWAP_TWO(): Intercambia dos bytes (sólo internamente).
*****************************************************************/

void SWAP_TWO( char *chr1, char *chr2 )
{
        char tmp;

        tmp = *chr1;
        *chr1 = *chr2;
        *chr2 = tmp;
}
/*****************************************************************
        SWAP(): Modifica el orden de bytes de una variable.
*****************************************************************/

void SWAP( void *pcr, int size )
{
        char *chr;

        chr = (char *)ptr;

        switch( size )
        {
                case 2:
                        SWAP_TWO( chr, chr+1 );
                        break;

                case 4:
                        SWAP_TWO( chr, char+3 );
```

```
                SWAP_TWO( chr+1, chr+2 );
                break;

        case 8:
                SWAP_TWO( chr, chr+7 );
                SWAP_TWO( chr+1, chr+6 );
                SWAP_TWO( chr+2, chr+5 );
                SWAP_TWO( chr+3, chr+4 );
                break;

        default:
ERROR( "SWAP(): Tamaño incorrecto" );
        }
}
```

La construcción fundamental del dispositivo se ilustra a partir de la figura.

Un objeto 1 se encuentra más cerca de un colimador multi-orificio 3 que de una superficie del detector 2. El colimador multi-orificio 3 presenta orificios 4 que desembocan desde ambos lados en forma de embudo en el colimador 3, para de este modo hacer posible un paso de cuantos gamma incidentes de modo oblicuo a través de los orificios. Las puntas 5 del colimador multi-orificio 3 están hechas de iridio. El resto de regiones del colimador multi-orificio 3 están hechas de wolframio. Los cuantos gamma 6 van a parar desde el objeto 1 a través de los orificios 4 sobre la superficie del detector 2. El objeto 1, de esta manera, se amplia en la superficie del detector 2. Entre los conos individuales que están conformados por medio de los cuantos gamma hay regiones de corte 7.

En la figura, los orificios 4 presentan distancia uniformes entre ellos. En una configuración de la invención, las distancias son irregulares.

REIVINDICACIONES

1. Procedimiento para la realización de un procedimiento tomográfico usando un dispositivo con un colimador multi-orificio (3) y un detector para el registro de cuantos gamma o fotones (6), que pasan a través del colimador multi-orificio, en el que la distancia entre un dispositivo de sujeción para un objeto (1) y el colimador multi-orificio (3) es menor que la distancia entre el colimador multi-orificio (3) y la superficie (2) del detector, en el que las distancias de los orificios individuales en el colimador multi-orificio (3) así como el tamaño y posición del objeto (1) se seleccionan de tal manera que los conos conformados por medio de los cuantos gamma o fotones (6) se cortan parcialmente en la superficie (2) del detector, y en el que se usa un procedimiento de reconstrucción según el que se asumen diferentes distribuciones de los radiofármacos en el objeto, a partir de ello se calculan resultados de medición que conseguirían las distribuciones asumidas y, como resultado de reconstrucción, se selecciona la distribución asumida cuyo resultado de medición calculado coincide mejor con el resultado de medición obtenido, realizándose la variante MLEM de la reconstrucción multi-orificio por medio de un algoritmo que tiene en cuenta la sensibilidad dependiente del lugar y la función de proyección dependiente del lugar de una apertura multi-orificio con orificios posicionados/inclinados arbitrariamente.

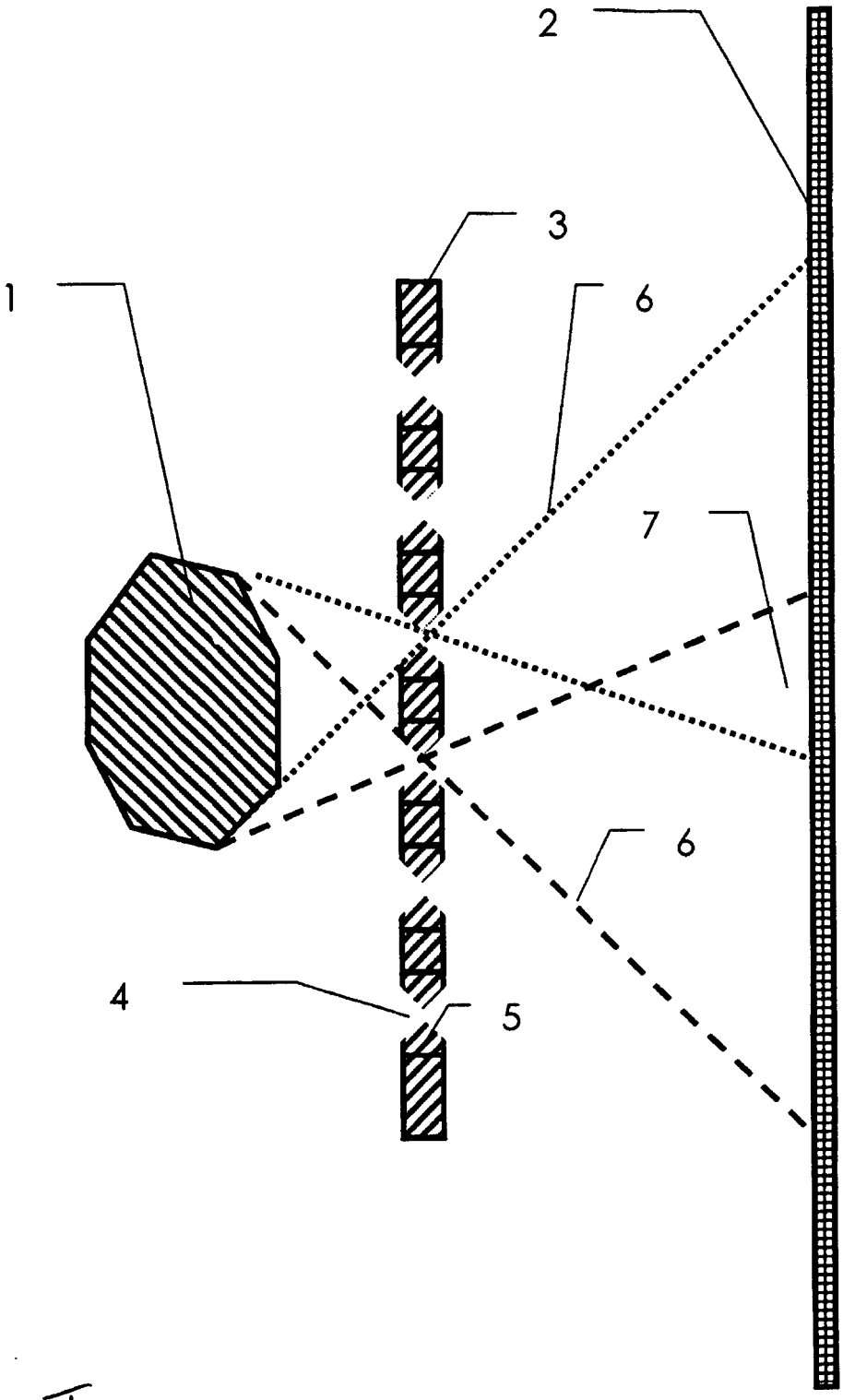

Fig. 1