```c
/* Multi Resolution Grid (MRG) : application to seismic tomography*/
/* Christophe Zaroli (2012) */

//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

#define BUFFMAX 80              // taille buffer lecture fichier donnees

//--------------------------------------------------------------------------------------------------

/* structure noeud pour creation arbre */

typedef struct noeud
       {
        int x; //abscisse coin bas gauche de la cellule
        int y; //ordonnee coin bas gauche de la cellule
        struct noeud *fils1;
        struct noeud *fils2;
        struct noeud *fils3;
        struct noeud *fils4;
       }Noeud;

//--------------------------------------------------------------------------------------------------

/* variables globales */

int g_N;
int g_M;

int g_l; // l=2^N : largeur de la cellule initiale (unite=pixel)
int g_L; // L=2^M : longeur de la cellule initiale (unite=pixel)

int g_nb;  // nb=l*L

int *g_tab_ray; // tableau des nombres de rais par pixel
int g_seuil; // seuil pour subdivision d'une cellule
int g_pmax; // profondeur de l'arbre maximale : min(N,M)
//--------------------------------------------------------------------------------------------------

/* prototypes des fonctions */

void aff_arbre(Noeud *, int);
void maj_tab_ray(int, int, int, int);
float projection_x(int , int , int, int , int );
float projection_y(int , int , int, int , int );
float ProduitScalaire(float [2], float [2]);
void lecture_data( char *);
void aff_tab_ray(int *, int, int);
int densite (int , int , int );
Noeud * allouerNoeud(int , int );
void developper(Noeud *, int );

//--------------------------------------------------------------------------------------------------
/* affichage des feuilles de l'arbre */

void aff_arbre(Noeud *a, int niv)
{

//indentation des fils par rapport au pere
   if(a!=NULL)
```

```
   {
        if(a->fils1 == NULL && a->fils2 == NULL && a->fils3 == NULL && a->fils4 ==
NULL )
            printf("%3d %3d %3d\n",niv, a->x, a->y);
        else
        {
            aff_arbre(a->fils1,niv+1);
            aff_arbre(a->fils2,niv+1);
            aff_arbre(a->fils3,niv+1);
            aff_arbre(a->fils4,niv+1);  }
   }
}
//-------------------------------------------------------------------------------------------------
/* mise a jour du tableau global g_tab_ray */

void maj_tab_ray(int x_s, int y_s, int x_r, int y_r)
{
int pixel; // indice du pixel dans le tableau g_tab_ray de 0 a g_nb-1
int x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D; // x: numero de colonne, y: numero de
ligne

/* equation du rai (droite) entre la source et le recepteur: ax+by+c=0 */

int coeff_a=y_r-y_s;
int coeff_b=x_s-x_r;
int coeff_c=y_s*x_r-x_s*y_r;

int FOUND;

for(pixel=0;pixel<g_nb;pixel++)
   {
    x_A=pixel%g_L;
    y_A=pixel/g_L;

    x_B=x_A+1;
    y_B=y_A;

    x_C=x_A+1;
    y_C=y_A+1;

    x_D=x_A;
    y_D=y_A+1;

    float x_Ap=projection_x(coeff_a, coeff_b, coeff_c, x_A, y_A);
    float y_Ap=projection_y(coeff_a, coeff_b, coeff_c, x_A, y_A);

    float x_Bp=projection_x(coeff_a, coeff_b, coeff_c, x_B, y_B);
    float y_Bp=projection_y(coeff_a, coeff_b, coeff_c, x_B, y_B);

    float x_Cp=projection_x(coeff_a, coeff_b, coeff_c, x_C, y_C);
    float y_Cp=projection_y(coeff_a, coeff_b, coeff_c, x_C, y_C);

    float x_Dp=projection_x(coeff_a, coeff_b, coeff_c, x_D, y_D);
    float y_Dp=projection_y(coeff_a, coeff_b, coeff_c, x_D, y_D);

    float a[2]={x_A-x_Ap, y_A-y_Ap};
    float b[2]={x_B-x_Bp, y_B-y_Bp};
    float c[2]={x_C-x_Cp, y_C-y_Cp};
    float d[2]={x_D-x_Dp, y_D-y_Dp};

     FOUND=0;

    if (ProduitScalaire(a,b)<=0.) FOUND=1;
       else if (ProduitScalaire(a,c)<=0.) FOUND=1;
             else if (ProduitScalaire(a,d)<=0.) FOUND=1;
```

```
    if (FOUND==1)
      g_tab_ray[pixel]=g_tab_ray[pixel]+1;
   }

}
//-------------------------------------------------------------------------------------------------------
/* abscisse de la projection d'un point sur la droite (rai) */

float projection_x(int a, int b, int c, int x, int y)
{
float x_p;
return x_p=(float) ( (pow(b,2)*x-a*b*y-a*c)/(pow(a,2)+pow(b,2)) );
}

//-------------------------------------------------------------------------------------------------------
/* ordonnee de la projection d'un point sur la droite (rai) */

float projection_y(int a, int b, int c, int x, int y)
{
float y_p;
return y_p= (float) ( (-a*b*x+pow(a,2)*y-b*c)/(pow(a,2)+pow(b,2)) );
}

//-------------------------------------------------------------------------------------------------------
/* produit scalaire entre deux vecteurs de dim 2 */

float ProduitScalaire(float X[2], float Y[2])
{
    return X[0]*Y[0]+X[1]*Y[1];
}
//-------------------------------------------------------------------------------------------------------
/* lecture des donnees du fichier en entree (couples source-station) */
/* et affectation des variables globales g_N et g_M, et construction du tableau
g_tab_ray */

void lecture_data( char * filename)
{
FILE *fichier=NULL;
char buff[BUFFMAX + 1];
int a,b,c,d;
int line=0;

//   printf("nom fichier = %s \n", filename);
   fichier=fopen(filename,"r+");

   if (fichier != NULL)
   {
       // printf("\nOn peut lire et ecrire dans le fichier\n");

       while(fgets(buff,BUFFMAX,fichier) != NULL)
        {
            sscanf(buff, "%d %d %d %d", &a,&b,&c,&d);
        //printf("\n %d %d %d %d", a,b,c,d);

            if( line > 0 )
                maj_tab_ray(a,b,c,d);
            else if ( line == 0 )
            {
                line=line+1;
                g_N = a;
                g_M = b;

                g_pmax = ( g_N<g_M ? g_N : g_M);

                g_L = pow(2,g_M) ;
```

```
                g_l = pow(2,g_N) ;
                g_nb = g_L * g_l;
                g_tab_ray = (int *) malloc( g_nb * sizeof (int) );
            }
         }
        fclose(fichier);
        //aff_tab_ray(g_tab_ray,g_l,g_L);
    }
    else
    {
        printf("\nImpossible d'ouvrir le fichier\n");
    }
}
//-------------------------------------------------------------------------------------------------
/* affichage du tableau g_tab_ray pour verification */

void aff_tab_ray(int *t, int dim1, int dim2)
{
int i, j;
printf("\n Affichage tableau g_tab_ray(%d, %d)\n\n", dim1, dim2);
for(i=0;i<dim1;i++) //lignes
   {
    for(j=0;j<dim2;j++) //colonnes
       {
        printf("%3d ", *(t+i*dim2+j));
       }
    printf("\n");
   }
}
//-------------------------------------------------------------------------------------------------
/* Calcule et retourne le nombre de rais (densite) passant par une cellule definie
par son point bas gauche (x,y) et le niveau de profondeur (p) dans l'arbre */

int densite (int x, int y, int p)
{
int s=0, i, j, n1, n2, n3;
n1=pow(2,g_N-p);
n2=pow(2,g_M-p);
n3=g_L;

for(i=y; i<y+n1; i++)
   {
    for(j=x; j<x+n2; j++)
       {
        s+=*(g_tab_ray+i*n3+j);
       }
   }
return s;
}
//-------------------------------------------------------------------------------------------------
/* alloue un nouveau noeud et initialise les coordonnes du point bas gauche de la
cellule */

Noeud * allouerNoeud(int x, int y)
{
Noeud *p;
p=(Noeud *)malloc(sizeof(Noeud));

p->x = x;
p->y = y;
p->fils1 = NULL;
p->fils2 = NULL;
p->fils3 = NULL;
p->fils4 = NULL;

return p;
```

```
}
//-------------------------------------------------------------------------------------------------
/* construit l'arbre de maillage de la cellule definie par le noeud a et par la
profondeur p */

void developper(Noeud *a, int p)
{
if( p+1 < g_pmax && densite(a->x, a->y, p) >= g_seuil)
 {
  //printf("\ndensite=%d , seuil=%d\n",densite(a->x, a->y, p),g_seuil);
  // on subdivise dans le sens trigo
  int n1, n2;
  n1=pow(2,g_N-p)/2; // demi largeur pour profondeur p+1
  n2=pow(2,g_M-p)/2; // demi longueur pour profondeur p+1

  a->fils1=allouerNoeud(a->x,a->y);
  a->fils2=allouerNoeud(a->x+n2,a->y);
  a->fils3=allouerNoeud(a->x+n2,a->y+n1);
  a->fils4=allouerNoeud(a->x,a->y+n1);

  // on rappelle la fonction pour chacun des fils (recursif)
  developper(a->fils1,p+1);
  developper(a->fils2,p+1);
  developper(a->fils3,p+1);
  developper(a->fils4,p+1);
 }
}
//-------------------------------------------------------------------------------------------------
// programme principal

void main(int argc, char **argv)
{
//srand(time(NULL));
Noeud *arbre=allouerNoeud(0,0); // arbre initial correspondant a une seule cellule
//printf("\n *** Debut Traitement (Multi-Resolution Grid) ***\n");
g_seuil = atoi(argv[2]);
//printf("seuil=%d",g_seuil);
lecture_data(argv[1]);
developper(arbre,0);
aff_arbre(arbre,0);
}
//-------------------------------------------------------------------------------------------------
```