```
/* This program was first created by P.Kotsas on January 25 1994 for
   his master's thesis on volume based rigid MR image registration.  The copyright of
   this work belongs to the department of Musculoskeletal radiology of the Cleveland
   Clinic Foundation , 9500 Euclid Avenue , Cleveland , Ohio 44195 . I was later
   extended  during October-November 2005 for 2d rigid registration using the 1d
   binary projections */



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

#define  IMAGE  short *
#define DIM 256
#define DX 53
#define DY 231
#define DX1 44
#define DY1 243

int flaga45;
short *a45,*axline,*ayline;  /*matrixes for binary projections of reference image*/

/* This program registers two RAW MR images . The options of this program are :



   1. REGISTER A RAW MR IMAGE TO ANOTHER RAW MR IMAGE

 */



int main(void)

{
  FILE *fpointer_a,*fpointer_b,*pf,*pf1,*pf2,*pf3;
  IMAGE a; /*reslice image*/
  IMAGE b; /*reference image*/
  short *abyte, *bbyte,*maska,*maskb; /*mask matrixes, masks created using slicer
©MIT*/
  char dummy[2],id1[512],id2[512],id3[512],id4[512];
  int dialog,i,j;
  short read_impact;
  double phi_xy,move_x,move_y;
  float move_x1,phi_xy1,move_y1;
  char char_flag;
```

```
extern int
register_images(IMAGE,IMAGE,int,int,double,double,double,double,double,double);

extern int read_sp_impact(char[],IMAGE,int); /*not used in case of RAW data*/

/*Memory allocation*/

a45=(short *)malloc(DIM*DIM*sizeof(short));
axline=(short *)malloc(2*DIM*sizeof(short));
ayline=(short *)malloc(DIM*8*sizeof(short));

maska=(short *)malloc(DIM*DIM*sizeof(short));
maskb=(short *)malloc(DIM*DIM*sizeof(short));
/*Initial conditions*/

read_impact=0;
phi_xy=0;
move_x=0;
move_y=0;
flaga45=0;



/* Dialogue in order to define what kind of images will be registered */



  printf("FOR RUNNING THIS PROGRAM YOU OBVIOUSLY WANT TO
REGISTER TWO IMAGE DATA SETS\n");
  printf("THE OPTIONS OF THIS PROGRAM ARE :\n");

  printf("1. REGISTER A RAW  MR IMAGE TO ANOTHER RAW MR
IMAGE\n");

  printf("0. NONE OF THE ABOVE . JUST EXIT THE PROGRAM\n");

  printf("THE DIMENSIONS OF THE SCANS SHOULD BE 256x256 \n");
  printf("\n");
  printf("GIVE ME YOUR CHOICE [0,1]...:");
  fscanf(stdin,"%d",&dialog);


  switch(dialog)
  {

    case 1 : read_impact=2;break;
    case 0 : printf("OK ... Bye !\n");exit(0);
    default  : printf("This option is not ready yet . Bye.\n");exit(0);
```

```
  }


/*Get the names of the image files */
  gets(dummy);
  printf("GIVE ME THE NAMES FOR THE TWO SETS OF IMAGE DATA : \n");


if(read_impact==2)
   {
   printf("IMAGE ONE(reference): ");
   gets(id1);
   printf("\n");
   printf("MASK ONE(reference): ");
   gets(id3);
   printf("\n");
   printf("IMAGE TWO(reslice): ");
   gets(id2);
   printf("\n");
   printf("MASK TWO(reslice): ");
   gets(id4);
   printf("\n");
   }



  /* Ask  for the initial estimates of the user  */


   printf("DO YOU WANT TO GIVE SOME INITIAL ESTIMATES ABOUT THE
ROTATION \n");
   printf("AND THE TRANSLATION NEEDED TO (DE)REGISTER THE IMAGES
?(if no \n");
   printf("the initial estimates will be considered  zerro). [y/n]: ");
   scanf("%c",&char_flag);
    printf("\n");

   if(char_flag=='y' || char_flag=='Y'){
    printf("GIVE ME THE INITIAL XY PLANE ROTATION ANGLE \n");
    scanf("%f",&phi_xy1);
    printf("GIVE ME THE INITIAL X TRANSLATION(pixels) \n");
    scanf("%f",&move_x1);
    printf("GIVE ME THE INITIAL Y TRANSLATION(pixels) \n");
    scanf("%f",&move_y1);}

phi_xy=(double)phi_xy1;
move_x=(double)move_x1;
move_y=(double)move_y1;
```

```
/*Allocate memory for the image buffers */

abyte=(short *)malloc(DIM*DIM*sizeof(short));
bbyte=(short *)malloc(DIM*DIM*sizeof(short));

a=(short *)malloc(DIM*DIM*sizeof(short));
b=(short *)malloc(DIM*DIM*sizeof(short));

pf = fopen(id1, "rb");
pf1=fopen(id2,"rb");
pf2=fopen(id3,"rb");
pf3=fopen(id4,"rb");


/* input  the images, masks and apply masks*/




fread(a, sizeof(short), DIM*DIM, pf1);
fread(b, sizeof(short), DIM*DIM, pf);
fread(maska,sizeof(short),DIM*DIM,pf3);
fread(maskb,sizeof(short),DIM*DIM,pf2);




for (i=0;i<256;i++)
for(j=0;j<256;j++)
{abyte[i*DIM+j]=0;
bbyte[i*DIM+j]=0;
}


for(i=DX1;i<DY1;i++)
for(j=DX;j<DY;j++)
{
if(maska[i*DIM+j]!=0)
 abyte[i*DIM+j]=b[i*DIM+j];
if(maskb[i*DIM+j]!=0)
 bbyte[i*DIM+j]=a[i*DIM+j];
 }


for (i=0;i<256;i++)
for(j=0;j<256;j++)
{a[i*DIM+j]=bbyte[i*DIM+j];
 b[i*DIM+j]=abyte[i*DIM+j];
```

```
 }

  /* Register  the two images  */

register_images(b,a,1,1,phi_xy,0,0,move_x,move_y,0);


  /* Dialogue  for the output images */


 printf("\n GIVE ME THE NAME OF THE OUTPUT FILES \n");

  if(read_impact==2)
  {
   printf("RESLICE IMAGE FILE: ");
   gets(id1);
   printf("\n");
   printf("REFERENCE IMAGE FILE: ");
   gets(id2);
    }


/*Write the images as raw files and exit*/

fpointer_a=fopen(id1,"wb");
fwrite(a,2,DIM*DIM,fpointer_a);
fclose(fpointer_a);
fpointer_b=fopen(id2,"wb");
fwrite(b,2,DIM*DIM,fpointer_b);
fclose(fpointer_b);
fclose(pf);
fclose(pf1);
fclose(pf2);
fclose(pf3);

free(maska);
free(maskb);
free(abyte);
free(bbyte);
free(a);
free(a45);
free(axline);
free(ayline);
free(b);
exit(0);
```

```
}


/*This routine was first created by P.Kotsas on January 22 1994 for his master thesis
   The copyright of this work belongs to the Clevelad Clinic Foundation ,
   9500 Euclid Avenue , Cleveland , Ohio 44195. It was later extended for 2d rigid
   registration using 1d binary projections*/



#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
#include <time64.h>
#include <time.h>

#define IMAGE short *
#define PI 3.141592653589793
#define DIM 256
#define DIM2 256
#define DIV 1
#define SAFETY -2
#define MEDIAN 25
#define MEDIAN2 12

/*The routine register_images registers two mri images . It is
  called by the register_mr_images.c program  */

double cent_x_origa, cent_y_origa;
double fimax,fimin,fjmax,fjmin;
double s_rot_xy, s_trans_x,s_trans_y;

int register_images(IMAGE a ,IMAGE b , int linta,int lintb , double rot_xy, double
rot_yz , double rot_zx , double trans_x , double trans_y , double trans_z )

/* IMAGE a ;
   IMAGE b;
   int linta;  /*number of slices=1*/
   int lintb;  /*number of slices =1*/
   float rot_xy;
   float rot_yz;
   float rot_zx;
   float trans_x;
   float trans_y;
   float trans_z;
*/
```

```
{

  extern double
compute_deriv_zerro(IMAGE,IMAGE,IMAGE,double,double,int,int,int,double,doubl
e,double);
  extern int
geom_trans_volume(IMAGE,IMAGE,int,double,double,double,double,double,double
,double,double,double,int,int,int,int,int,int);




   IMAGE c, *flaga,*flagb,*b2,*a2;
   IMAGE d;
   double rescent_x,rescent_y;
   double rot_xyinit,trans_xinit,trans_yinit;
   double cent_x_orig,cent_y_orig;
   char dummy[2];
   int i,j,flag1,order[220],time,s_new,thresh1,thresh2;
   double cent_x,cent_y,cent_z;
   double a_x,a_y,b_x,b_y,a_c,b_c;
   int stop_xy,stop_x,stop_y;
   time64_t *tp1, *tp2;
   time64_t ctime0,ctime1;
   clock_t ticksnow1, ticksnow2;
   float tused;
   short s1[30],s2[30],*hista,*histb,sa,sb;
   int nm0,k,l,suma,nm02,c1,c2,nm1,nm2,kk,ll;
   double m1,m2,m1old,m2old,t0a,sm0,sm02,sm1,sm2,*ara,*arb;
   short help,max;

/*Initial conditions   */


a_x=a_y=b_x=b_y=a_c=b_c=0;
stop_xy=stop_x=stop_y=0;
  cent_z=0;
  cent_x=0;
  cent_y=0;
  s_rot_xy=0;
  s_trans_x=0;
  s_trans_y=0;


time=0;
/*order of transformations 1->xy rotation, 4->x translation 5->y translation*/
for(i=0;i<200;i+=3)
 { order[i]=5;order[i+1]=4;order[i+2]=1;}
```

```
/*initialize transformations*/

rot_xyinit=rot_xy;
trans_xinit=trans_x/DIV;
trans_yinit=trans_y/DIV;



/*Allocate memory for the two reslice volume transformation buffers*/


d=(short *)malloc(DIM2*DIM2*sizeof(short));
if(d==NULL){
   printf("I cannot allocate memory for the reslice image \n");
   exit(-1);
   }


 c=(short *)malloc(DIM2*DIM2*sizeof(short));
if(c==NULL){
   printf("I cannot allocate memory for the reslice image \n");
   exit(-1);
   }

flaga=(short *)malloc(DIM*DIM*sizeof(short));
flagb=(short *)malloc(DIM*DIM*sizeof(short));
tp1=(time64_t *)malloc(1*sizeof(time64_t));
tp2=(time64_t *)malloc(1*sizeof(time64_t));
ara=(double *)malloc(4096*sizeof(double));
arb=(double *)malloc(4096*sizeof(double));
hista=(short *)malloc(4096*sizeof(short));
histb=(short *)malloc(4096*sizeof(short));
b2=(short *)malloc(DIM2*DIM2*sizeof(short));
a2=(short *)malloc(DIM2*DIM2*sizeof(short));




/*thresholding to eliminate noise*/




/*Automated thresholding*/

suma=0;
for (i=0;i<DIM;i++)
```

```
for (j=0;j<DIM;j++)

{
 suma+=a[i*DIM+j];
 }

 t0a=(int)((double)suma/(DIM*DIM));
 printf("%f \n",t0a);

 m1=20000;
 m2=20000;
 m1old=0;
 m2old=0;
 nm0=0;
 sm0=0;
 sm02=0;
 nm02=0;
 for (i=0;i<DIM;i++)
      for (j=0;j<DIM;j++)
        {
          if (a[i*DIM+j]<=t0a) {sm0+=a[i*DIM+j]; nm0++;}
          else {sm02+=a[i*DIM+j]; nm02++;}

        }
 c1=(int)((double)sm0/(double)nm0);
 c2=(int)(0.2*(double)sm02/(double)nm02);
 printf("%d %d \n",c1,c2);

 while (abs(m1-m1old)>c1 || abs(m2-m2old)>c2)
           { nm1=0;nm2=0;sm1=0;sm2=0;
            for (i=0;i<DIM;i++)
      for (j=0;j<DIM;j++)
        {
          if (a[i*DIM+j]>t0a) {sm2+=a[i*DIM+j]; nm2++;}
                          else
                    {sm1+=a[i*DIM+j]; nm1++;}
        }
      printf("%f %f %d %d\n", sm1,sm2,nm1,nm2);
      m1old=m1; m2old=m2;
                 m1=(double)((double)sm1/(double)nm1);
                 m2=(double)((double)sm2/(double)nm2);
     t0a=(double)(0.5*((double)m1+(double)m2));

  }


thresh1=t0a;
```

```
 suma=0;
for (i=0;i<DIM;i++)
 for (j=0;j<DIM;j++)

 {
  suma+=b[i*DIM+j];
  }

  t0a=(int)((double)suma/(double)(DIM*DIM));

  m1=2000;
  m2=2000;
  m1old=0;
  m2old=0;
  nm0=0;
  sm0=0;
  sm02=0;
  nm02=0;
  for (i=0;i<DIM;i++)
       for (j=0;j<DIM;j++)
         {
           if (b[i*DIM+j]<=t0a) {sm0+=b[i*DIM+j]; nm0++;}
           else {sm02+=b[i*DIM+j]; nm02++;}

         }
  c1=(int)((double)sm0/(double)nm0);
  c2=(int)(0.2*(double)sm02/(double)nm02);


  while (abs(m1-m1old)>c1 || abs(m2-m2old)>c2)
             { nm1=0;nm2=0;sm1=0;sm2=0;
              for (i=0;i<DIM;i++)
       for (j=0;j<DIM;j++)
         {
           if (b[i*DIM+j]>t0a) {sm2+=b[i*DIM+j]; nm2++;}
                              else
                        {sm1+=b[i*DIM+j]; nm1++;}
         }
       m1old=m1; m2old=m2;
                    m1=((double)sm1/(double)nm1);
                    m2=((double)sm2/(double)nm2);
      t0a=(int)(0.5*((double)m1+(double)m2));

   }


thresh2=t0a;
```

```
printf("%d  %d \n", thresh1,thresh2);

ticksnow1=clock();



 for(i=0;i<DIM;i++)
  for(j=0;j<DIM;j++)
   {
    if(a[i*DIM+j]<thresh1) a[i*DIM+j]=0;
    if(b[i*DIM+j]<thresh2) b[i*DIM+j]=0;
   }




/*Compute centroids of the reference and reslice image*/


for(i=0;i<DIM;i++)
 for(j=0;j<DIM;j++)
   {
  if(a[i*DIM+j]!=0) {b_x+=j;b_y+=i;b_c++;}
}


cent_x=(double)(b_x/b_c);
cent_y=(double)(b_y/b_c);




cent_x_origa=(double)(cent_x/DIV);
cent_y_origa=(double)(cent_y/DIV);




for(i=0;i<DIM;i++)
 for(j=0;j<DIM;j++)
   {
              flaga[i*DIM+j]=0;
              flagb[i*DIM+j]=0;
  if(b[i*DIM+j]!=0) {b_x+=j;b_y+=i;b_c++;}
}

cent_x=(double)(b_x/b_c);
```

```
cent_y=(double)(b_y/b_c);



cent_x_orig=(double)(cent_x/DIV);
cent_y_orig=(double)(cent_y/DIV);
cent_x=cent_x_orig;
cent_y=cent_y_orig;

/*Compute the neighbouring to zero pixels*/
for(i=1;i<DIM-1;i++)
  for(j=1;j<DIM-1;j++)
            if (a[i*DIM+j]!=0)
                    if (a[(i-1)*DIM+j]!=0 && a[(i+1)*DIM+j]!=0 && a[i*DIM+j-1]!=0
&& a[i*DIM+j+1]!=0)
                     flaga[i*DIM+j]=99;


for(i=1;i<DIM-1;i++)
  for(j=1;j<DIM-1;j++)
            if (b[i*DIM+j]!=0)
                    if (b[(i-1)*DIM+j]!=0 && b[(i+1)*DIM+j]!=0 && b[i*DIM+j-1]!=0
&& b[i*DIM+j+1]!=0)
                     flagb[i*DIM+j]=99;



for(i=0;i<DIM;i++)
  for(j=0;j<DIM;j++)
            { if(flaga[i*DIM+j]!=0) a[i*DIM+j]=0;
     if(flagb[i*DIM+j]!=0) b[i*DIM+j]=0;
  }



ll=-1;
for(i=0;i<DIM;i+=DIV)
 {kk=-1;ll++;
 for(j=0;j<DIM;j+=DIV)
            {
             kk++;
             if(flaga[i*DIM+j]!=0) a2[ll*DIM2+kk]=0; else
a2[ll*DIM2+kk]=a[i*DIM+j];
     if(flagb[i*DIM+j]!=0) b2[ll*DIM2+kk]=0; else b2[ll*DIM2+kk]=b[i*DIM+j];
  }

}

trans_x/=DIV;
trans_y/=DIV;
```

```
/* START MAIN LOOP */

while(time<16)

{


  s_rot_xy+=rot_xy;
  s_trans_x+=trans_x;
  s_trans_y+=trans_y;

  time++;









if(stop_xy==SAFETY&&stop_x==SAFETY&&stop_y==SAFETY)
  break;






/*decide which transformation you will adjust */

   flag1=order[time-1];

/*compute the minimum variance point for the chosen transformation */


/* adjust xy rotation*/

  if(flag1==1&&stop_xy>SAFETY) {

  rot_xy=compute_deriv_zerro(a2,b2,d,-0.1,0.1,50,1,1,cent_x,cent_y,0);

  if(fabs(rot_xy)<=1) stop_xy+=-1;
  trans_x=0.0;
  trans_y=0.0;


  }
```

```
/* adjust translation x*/

if(flag1==4&&stop_x>SAFETY) {

  trans_x=compute_deriv_zerro(a2,b2,d,-0.2,0.2,50,4,1,cent_x,cent_y,0);


  if(fabs(trans_x)<=1) stop_x+=-1;



  rot_xy=0.0;
  trans_y=0.0;


  }

/*adjust translation y*/

if(flag1==5&&stop_y>SAFETY) {

  trans_y=compute_deriv_zerro(a2,b2,d,-0.2,0.2,50,5,1,cent_x,cent_y,0);

  if(fabs(trans_y)<=1) stop_y+=-1;



  trans_x=0.0;
  rot_xy=0.0;


  }


if(flag1==1&&stop_xy==SAFETY){


  trans_x=0.0;
  trans_y=0.0;

  }



if(flag1==4&&stop_x==SAFETY){
```

```
  rot_xy=0.0;



  trans_y=0.0;

  }

if(flag1==5&&stop_y==SAFETY){
  rot_xy=0;

  trans_x=0;


  }

}  /*while*/

ticksnow2=clock();
tused=(float)(ticksnow2-ticksnow1)/CLOCKS_PER_SEC;
printf(" %f clock time \n", tused);

printf("\n\n\n\n");
printf("I applied the following transformations :\n");
printf("ROTATION ON XY PLANE :%f degrees \n",s_rot_xy/*-rot_xyinit*/);
printf("TRANSLATION ALONG X AXIS : %f pixels \n",s_trans_x/*-trans_xinit*/);
printf("TRANSLATION ALONG Y AXIS : %f pixels \n",s_trans_y/*-trans_yinit*/);

gets(dummy);


/*Free allocated memory */
free(c);
free(d);
free(flaga);
free(flagb);
free(tp1);
free(tp2);
free(ara);
free(arb);
free(hista);
free(histb);
free(a2);
free(b2);

return(0);

}
```

```
/*This routine was first created by P.Kotsas on January 24 1994. It was later extended
during Oct-Nov 2005 for 2d rigid registration using 1d binary projections. */


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


#define PI  3.141592653589793
#define IMAGE short *


float compute_deriv_zerro(IMAGE a_im,IMAGE c_im,IMAGE d_im,double
a,double b,int n,int flag,int lintb,double cent_x,double cent_y,double cent_z)

{
  extern  double variance(
IMAGE,IMAGE,IMAGE,int,int,double,double,double,double);
  double cheb_value(double,double *,int);
  int j,k;

  double f[3600];
  double cheb_variance[30];

  double fac,bpa,bma,y;

  double der_zerro,fact,min,k_min,kr;
  double sum=0.0;

  FILE *fvar;
  char filename[82];



 if(flag==1||flag==2||flag==3){a*=360;b*=360;fact=360;n=5;}
 else {a*=180;b*=180;fact=180;n=5;}



/* Compute chebyshev approximation of the variance of the ratio */


  /*n=5;*/ /*NUMBER OF CHEBYSHEV POINTS*/
```

```
 bma=0.5*(b-a);
 bpa=0.5*(b+a);
 /*printf("%f  ",bma);*/
 j=-1;
 for(k=0;k<n;k++) {
 j++;
 y=cos(PI*(k+0.5)/n);
 /*printf("%d FLAG %f VALUE \n", flag,y*bma+bpa);*/
 f[j]=variance(a_im,c_im,d_im,1,flag,y*bma+bpa,cent_x,cent_y,cent_z);
/* printf("%f f[j]\n",f[j]);*/
  }


 fac=2.0/n;


 for(j=0;j<n;j++){
  sum=0.0;
 for(k=0;k<n;k++)
  sum+=f[k]*cos(PI*j*(k+0.5)/n);

cheb_variance[j]=fac*sum;




 }

min=100000000;

/*printf("give me the name of the file for variance data\n");
gets(filename);*//*
fvar=fopen(filename,"wb");
*/
 for(k=0;k<=320;k++) {
    y=-1+(float)(k*0.025)/4.0;    /*2*(float)k/90.0*/;
    f[k]=cheb_value(y,cheb_variance,n);

 /*  printf("%f %f \n",y*bma+bpa,f[k]);
*/
if(f[k]<min){min=f[k]; k_min=y*bma+bpa;}
  }

/*fclose(fvar);*/

/*printf("at %f minimum %f\n",k_min,min);

*/
```

```
/*printf("%f \n",k_min);
*/
   return(k_min);
   }




  double cheb_value( double x , double *cheb,int n)
/*used internally by compute_deriv_zerro routine */
  {
   double value ;
   double t[1000];
   int i;



   t[0]=1;t[1]=x;value=0;

   for(i=2;i<n;i++)
    t[i]=2*x*t[i-1]-t[i-2];

   for(i=0;i<n;i++)
    value+=cheb[i]*t[i];

   value-=0.5*cheb[0];

   return(value);
   }
```

```
/*This program was created by P.Kotsas on January 25 1994 for his master
  thesis . The copyright belongs to the Cleveland Clinic Foundation . It was later
extended for 2d rigid registration using 1d binary projections */

#include <stdio.h>
#include <malloc.h>
#include <math.h>

#define IMAGE short *
#define PI 3.141592653589793
#define DIM 256
```

```
#define DIM4 127

double
aimin,aimax,ajmin,ajmax,ai45min,ai45max,aj45min,aj45max,aim45min,aim45max,m
ax,ajm45min,ajm45max;

double variance(IMAGE a,IMAGE c,IMAGE d,int lintb,int flag , double value,double
cent_x, double cent_y, double cent_z)

{

 FILE *ratpf,*conpf;
 double coeff;
 int a1,d1,j_num;
 int i,j,k,ii,jj,kk,lint2,ld1;
 double mean,var,svar,var_ratio,s1;
 extern int
geom_trans_volume(IMAGE,IMAGE,int,double,double,double,double,double,double
,double,double,double,int,int,int,int,int,int);
 short /* *axline, *ayline,*/ *bxline, *byline, *d45, /**a45,*/*am45,*dm45;
 double value1;
 double xy1,tx1,ty1;
 extern int flaga45;
 extern short *a45, *axline, *ayline;
 extern double cent_x_origa, cent_y_origa;
 extern double fimin,fimax,fjmin,fjmax;
 extern int nearint(double);
 int help;
 extern double s_rot_xy,s_trans_x,s_trans_y;

 double
bimin,bimax,bjmin,bjmax,bi45min,bi45max,bj45min,bj45max,bjm45min,bjm45max,
bim45min,bim45max;


bxline=(short *)malloc(2*DIM*sizeof(short));
byline=(short *)malloc(DIM*8*sizeof(short));

d45=(short *)malloc(DIM*DIM*sizeof(short));
am45=(short *)malloc(DIM*DIM*sizeof(short));
dm45=(short *)malloc(DIM*DIM*sizeof(short));




/*transform the reslice volume */

flaga45++;
```

```
if(flag==1)
{
xy1=s_rot_xy+value;


geom_trans_volume(c,d,1,xy1,0,0,cent_x,cent_y,0,s_trans_x,s_trans_y,0,0,0,0,DIM-
1,DIM-1,0);

bimin=fimin;
bimax=fimax;
bjmin=fjmin;
bjmax=fjmax;
}

if(flag==4)
{
tx1=s_trans_x+value;

geom_trans_volume(c,d,1,s_rot_xy,0,0,cent_x,cent_y,0,tx1,s_trans_y,0,0,0,0,DIM-
1,DIM-1,0);

bimin=fimin;
bimax=fimax;
bjmin=fjmin;
bjmax=fjmax;

}

if(flag==5)
{
ty1=s_trans_y+value;

geom_trans_volume(c,d,1,s_rot_xy,0,0,cent_x,cent_y,0,s_trans_x,ty1,0,0,0,0,DIM-
1,DIM-1,0);

bimin=fimin;
bimax=fimax;
bjmin=fjmin;
bjmax=fjmax;

}

/*compute projection minima-maxima*/
value1=-45.0;

if (flaga45==1)
{geom_trans_volume(a,a45,1,value1,0,0,cent_x_origa,cent_y_origa,0,0,0,0,0,0,0,DI
M-1,DIM-1,0);
```

```
 ai45min=fimin;
 ai45max=fimax;
 aj45min=fjmin;
 aj45max=fjmax;
 geom_trans_volume(a,a45,1,0,0,0,cent_x_origa,cent_y_origa,0,0,0,0,0,0,0,DIM-
1,DIM-1,0);

 aimin=fimin;
 aimax=fimax;
 ajmin=fjmin;
 ajmax=fjmax;
 }



if (flag==1)
{
xy1=s_rot_xy+value+value1;
geom_trans_volume(c,d45,1,xy1,0,0,cent_x,cent_y,0,s_trans_x,s_trans_y,0,0,0,0,DI
M-1,DIM-1,0);


}
bi45min=fimin;
bi45max=fimax;
bj45min=fjmin;
bj45max=fjmax;


/*compute binary projections*/


help=(int)(aimin);
if (flaga45==1)
{
for(i=0;i<2*DIM;i++)
if (i>=128+(int)(aimin) && i<=128+(int)(aimax))
 {
                    ayline[i]=255;
}
 else
  ayline[i]=0;


for(i=0;i<2*DIM;i++)
if (i>=128+(int)(ai45min) && i<=128+(int)(ai45max))
 {
          ayline[i+2*DIM]=255;
}
```

```
 else
  ayline[i+2*DIM]=0;


for(j=0;j<2*DIM;j++)
if (j>=128+(int)(ajmin) && j<=128+(int)(ajmax))
 {
            axline[j]=255;

 }
 else
  axline[j]=0;

}


for(i=0;i<2*DIM;i++)
if (i>=128+(int)(bimin) && i<=128+(int)(bimax))
 {
            byline[i]=255;
}
 else
  byline[i]=0;


for(j=0;j<2*DIM;j++)
if (j>=128+(int)(bjmin) && j<=128+(int)(bjmax))
 {
            bxline[j]=255;
}
 else
  bxline[j]=0;


for(i=0;i<2*DIM;i++)
if (i>=128+(int)(bi45min) && i<=128+(int)(bi45max))
 {
            byline[i+2*DIM]=255;

 }
 else
  byline[i+2*DIM]=0;



/*Compute variance using binary projections */


var=0;mean=0;j_num=0;
```

```
 for(i=0;i<2*DIM;i++)
  for(j=0;j<2*DIM;j++)
   {
   if (flag==1)
   {
   d1=/*bxline[j]+byline[i]*/byline[i+2*DIM]/*+byline[i+4*DIM]*/;
   a1=/*axline[j]+ayline[i]*/ayline[i+2*DIM]/*+ayline[i+4*DIM]*/;
   }
   if (flag==4 || flag==5)
   {
   d1=bxline[j]+byline[i]/*+byline[i+DIM]*//*+byline[i+2*DIM]*/;
   a1=axline[j]+ayline[i]/*+ayline[i+DIM]*//*+ayline[i+2*DIM]*/;

   }
   if(d1!=0&&a1!=0)
      {
       ld1=1;
       j_num++;
     s1=ld1*((double)d1/(double)a1)+ld1*((double)a1/(double)d1);
       var+=1;
      }
    if((a1!=0&&d1==0)||(a1==0&&d1!=0))
       {
       var+=100000000;
       j_num++;
        }


     }

var/=j_num;

var_ratio=var;

/*Free allocated memory and exit */
free(bxline);
free(byline);
free(d45);

return(var_ratio);


}
```

```
 /*This routine was first created by P.Kotsas on January 17 1994 for his master thesis
   The copyright of this work belongs to the Cleveland Clinic Foundation .
 It was later modified to be used for 2d rigid registration using 1d binary projections
*/


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

#define  VOLUME short *
#define  DIM0  256



/*The routine geom_trans_volume transforms the volume around (ic,jc,kc) for
  rotations rotxy,rotxz,rotyz and translations x,y,z around and along the three
  axes. It computes the minima and maxima for projection computation.  */

int geom_trans_volume(VOLUME a3,VOLUME b3,int lint,double rotxy1,double
rotyz1,double rotzx1,double jc,double ic,double kc,double x1,double y1,double z1,int
K1,int L1,int M1,int K2,int L2,int M2)


{
  int nearint(double);
  unsigned char *sa;
  int i,j,k,ii,jj,kk,l,ll,flag,num,skip;
  double PI=4.0*atan((double)1.0),xyres=1,sum;
  double
tx,ty,tz,ctx,cty,ctz,stx,sty,stz,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,q1,q2;
double
q11[DIM0],q21[DIM0],q3[DIM0],q4[DIM0],q5[DIM0],q6[DIM0],q7[DIM0],q8[DI
M0],q9[DIM0],qa[DIM0],qb[DIM0],st1,st2,st3,st;
  int DIM;
  double fii,fjj,fkk;
  extern double fimax,fimin,fjmax,fjmin;


DIM=256;


flag=0;
skip=1;



   for(k=0;k<lint;k++)
   for(i=0;i<DIM;i++)
```

```
   for(j=0;j<DIM;j++)
   {
            b3[k*DIM*DIM+i*DIM+j]=0;

   }

/*transform degrees to rads */

   tx=(double)rotyz1*PI/180.0;
   ty=(double)rotzx1*PI/180.0;
   tz=(double)rotxy1*PI/180.0;
   stx=sin(tx);
   sty=sin(ty);
   stz=sin(tz);
   ctx=cos(tx);
   cty=cos(ty);
   ctz=cos(tz);

     for(i=0;i<DIM;i++)
     {qa[i]=ctz*i;qb[i]=stz*i;}
       q1=-ctz*jc+stz*ic+x1+jc;
       q2=-stz*jc-ctz*ic+y1+ic;
             fimax=-DIM;
             fimin=2*DIM;
             fjmax=-DIM;
             fjmin=2*DIM;
        for(k=0;k<lint;k++)
        for(i=0;i<DIM;i++)
         for(j=0;j<DIM;j++)
         {
           if(a3[k*DIM*DIM+i*DIM+j]==0) continue;

            fjj=qa[j]-qb[i]+q1;
            fii=qb[j]+qa[i]+q2;
            fkk=0 /*k+z*/;
                     if (fjj>fjmax) fjmax=fjj;
                              if (fjj<fjmin) fjmin=fjj;
                              if (fii>fimax) fimax=fii;
                              if (fii<fimin) fimin=fii;

          }



/*printf("fimin %f fimax %f fjmin %f fjmax %f \n", fimin,fimax,fjmin,fjmax);
*/
```

```
return(0);

}
```

```
/* Function nearint computes the nearest integer to a double coordinate*/

int nearint(double orig)
{
 int near;

  if((orig-(int)orig)>0.5) near=(int)orig/*+1*/;
   else near=(int)orig;
  return(near);
 }
```