Subject: arbitrary rotation of 3-d arrays
Posted by D. Mattes on Thu, 10 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

hello idl users:
has anybody out there in idl-land written or seen code to apply arbitrary
rotations to 3-d arrays???

thanks in advance!

david mattes

Subject: Re: arbitrary rotation of 3-d arrays
Posted by David Foster on Fri, 11 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

morisset@my-deja.com wrote:
>
> The use of t3d will perform transformation on a coordinate
> cube, but will not 'rotate' the datas. It's like when you want to
> compare 2 images with one turned in respect to the other one. Or if you
> want to make a projection using total, but on an axis other than x or y.
> Then you have to use the ROT idl function to perform interpolation.
>
> I wrote a turn3d code (see below) that perform rotation of the datas
> in a 3D cube. It uses ROT slide by slide.
> It (seems to ;-) works also with vector fields.
> Hope it helps, send (bug) repports to (witout blanck):
> morisset @ astrsp-mrs.fr

Christophe -

The previous posts discussing the use of T3D were all assuming that
once the transformation of coordinates was performed, one would have
to then use interpolation to actually transform the data. At least in
my post I simply forgot to mention this important step. (oops!)

My main concern with your method is that it is using 2D techniques
to perform a 3D task, and I believe will invariably suffer from poorer
performance. Both methods will require large amounts of memory, but
in your method you make entire copies of the original data-set, and
three times! Often the data for such an operation will be huge, as
in the case of medical images. The performance penalties this copying
will incur, as well as the use of for loops to process each set of
2D images through the data, will make this method much slower than
the use of T3D and interpolation.

I would also argue that it would be less accurate, since you are performing interpolation three times in succession basically, once within each 2D plane, whereas the T3D method will transform the coordinates mathematically and then interpolate the original data once from those computed coordinates. Quite a different animal.

Dave Foster

```
>
>  ----------Cut here -------------------
>  function turn_3d,a_in,x_angle,y_angle,z_angle,RESIZE = resize,$
>         CONSERV = conserv,VERBOSE = verbose,HELP=help,_extra= _extra,vect=vect
>  ;+
>  ; NAME:
>  ;      turn_3d
>  ;
>  ; CALLING SEQUENCE:
>  ;      result = turn_3d(a,x_angle,y_angle,z_angle)
>  ;
>  ; PURPOSE:
>  ;      Rotate a 3D array. It applys the ROT IDL function to each
>  ;      2D sub_array of A. The computation is done in a 50% bigger
>  ;      cube to assure that nothing will be losed.
>  ;      If A is a structure, apply recurively turn_3d on all the
>  ;      tags of the structure.
>  ;
>  ; INPUT PARAMETERS:
>  ;      A =     The 3D array to be rotated. This array may be of any type
>  ;              exepted string. Can be a structure.
>  ;      X, Y, Z_ANGLE =
>  ;              3 angles of rotation in degrees CLOCKWISE.
>  ;      WARNING: in case of multiple rotation, the order is Z, X and Y.
>  ; KEYWORDS:
>  ;      VECT : Setting this keyword if A is a 3D vector field
>  ;      _extra will be passed to ROT:
>  ;
>  ;      RESIZE: Setting this keyword to resize the result to the maximum
>  ;              size (x,y or z-one) of A. The resizing is NOT a rebining,
>  ;              it extracts a 3D sub-array of the big 3D array in wich
>  ;              the computation is done.
>  ;              If A is a structure, RESIZE is set.
>  ;      CONSERVE: Setting this keyword to assure that
>  ;              total(result) = total(A).
>  ;
>  ;      VERBOSE: Setting this keyword will print the ratio of the
>  ;              sizes of the input array and the result. Works only if
>  ;              RESIZE not set. If A is a structure, will say what is
>  ;              being rotated.
```

```
> ;       HELP: print the calling sequence
> ;
> ; LIMITATIONS: They are those of ROT... For small dimensions arrays,
> ;       a rotation of +10deg followed by a rotation of -10deg will NOT
> ;       give you back the input data.
> ;
> ; BUGS: If A is a structure of arrays NON cubics (s(1) = s(2) = s(3),
> ;       then it crash!
> ;
> ; AUTHOR
> ;       Christophe MORISSET, 1997. morisset @ iagusp.usp.br
> ;
> ; HISTORY:
> ;       15-9-97 Post for me by D. Fanning on comp.lang.idl-pvwave
> ;       19-9-97 Add the HELP keyword
> ;       26-9-97 Add the possibiliy for A to be a structure
> ;               Suppretion of CUBIC keyword
> ;       28-4-98 pass all the _extra to rot
> ;       19-1-99 add the vector facility (and keyword)
> ;-

<code deleted>

--
```


~~~~~~ ~~~~~~~

David S. Foster Univ. of California, San Diego
Programmer/Analyst Brain Image Analysis Laboratory
foster@bial1.ucsd.edu Department of Psychiatry
(619) 622-5892 8950 Via La Jolla Drive, Suite 2240
La Jolla, CA 92037

~~~~~~ ~~~~~~~

Subject: Re: arbitrary rotation of 3-d arrays
Posted by morisset on Fri, 11 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

The use of t3d will perform transformation on a coordinate
cube, but will not 'rotate' the datas. It's like when you want to
compare 2 images with one turned in respect to the other one. Or if you
want to make a projection using total, but on an axis other than x or y.
Then you have to use the ROT idl function to perform interpolation.

I wrote a turn3d code (see below) that perform rotation of the datas
in a 3D cube. It uses ROT slide by slide.
It (seems to ;-) works also with vector fields.
Hope it helps, send (bug) repports to (witout blanck):

morisset @ astrsp-mrs.fr

----------Cut here ------------------

```
function turn_3d,a_in,x_angle,y_angle,z_angle,RESIZE = resize,$
 CONSERV = conserv,VERBOSE = verbose,HELP=help,_extra= _extra,vect=vect
;+
; NAME:
; turn_3d
;
; CALLING SEQUENCE:
; result = turn_3d(a,x_angle,y_angle,z_angle)
;
; PURPOSE:
; Rotate a 3D array. It applys the ROT IDL function to each
; 2D sub_array of A. The computation is done in a 50% bigger
; cube to assure that nothing will be losed.
; If A is a structure, apply recurively turn_3d on all the
; tags of the structure.
;
; INPUT PARAMETERS:
; A =  The 3D array to be rotated. This array may be of any type
;  exepted string. Can be a structure.
; X, Y, Z_ANGLE =
;  3 angles of rotation in degrees CLOCKWISE.
;       WARNING: in case of multiple rotation, the order is Z, X and Y.
; KEYWORDS:
;       VECT : Setting this keyword if A is a 3D vector field
; _extra will be passed to ROT:
;
; RESIZE: Setting this keyword to resize the result to the maximum
;  size (x,y or z-one) of A. The resizing is NOT a rebining,
;  it extracts a 3D sub-array of the big 3D array in wich
;  the computation is done.
;  If A is a structure, RESIZE is set.
; CONSERVE: Setting this keyword to assure that
;  total(result) = total(A).
;
; VERBOSE: Setting this keyword will print the ratio of the
;  sizes of the input array and the result. Works only if
;  RESIZE not set. If A is a structure, will say what is
;  being rotated.
; HELP: print the calling sequence
;
; LIMITATIONS: They are those of ROT... For small dimensions arrays,
; a rotation of +10deg followed by a rotation of -10deg will NOT
; give you back the input data.
;
; BUGS: If A is a structure of arrays NON cubics (s(1) = s(2) = s(3),
```

```
; then it crash!
;
; AUTHOR
; Christophe MORISSET, 1997. morisset @ iagusp.usp.br
;
; HISTORY:
; 15-9-97 Post for me by D. Fanning on comp.lang.idl-pvwave
; 19-9-97 Add the HELP keyword
; 26-9-97 Add the possibiliy for A to be a structure
;  Suppretion of CUBIC keyword
;       28-4-98 pass all the _extra to rot
;       19-1-99 add the vector facility (and keyword)
;-

   if keyword_set(help) then begin
      print,'function turn_3d,a,x_angle,y_angle,z_angle,INTERP =
interp,'+ $
       'MISSING = missing,PIVOT = pivot, RESIZE = resize,'+ $
       'CONSERV = conserv,VERBOSE = verbose,HELP=help,vect=vect'
      return,0
   endif


   if (size(a_in))(n_elements(size(a_in))-2) eq 8 then begin ; a is a
structure
      if keyword_set(verbose) then print,' turn_3d: structure'
      b = a_in
      names = tag_names(a_in)
      for i = 0,n_tags(a_in)-1 do begin
         if keyword_set(verbose) then print,'turning ',names(i)
         b.(i) = turn_3d(a_in.(i),x_angle,y_angle,z_angle, $
                         _extra=_extra,$
                         RESIZE = 1,CONSERV = conserv)
      endfor
      return,b

   endif                        ; case a is a structure

   if keyword_set(vect) then  begin
      if keyword_set(verbose) then print,' turn_3d: vector'
      a_out =  a_in

      if z_angle ne 0. then begin
         a_tmp =  a_out
         t3d,/reset,rotate=[0.,0.,z_angle]
         for i= 0,2 do a_out[*,*,*,i] = $
          a_tmp[*,*,*,0] * !p.t[i,0] + $
          a_tmp[*,*,*,1] * !p.t[i,1] + $
```

```
         a_tmp[*,*,*,2] * !p.t[i,2]
        for i= 0,2 do a_out[*,*,*,i] = turn_3d(a_out[*,*,*,i], $

0.,0.,z_angle,_extra=_extra, $

resize=resize,verbose=verbose)
      endif
      if x_angle ne 0. then begin
        a_tmp =  a_out
        t3d,/reset,rotate=[x_angle,0.,0.]
        for i= 0,2 do a_out[*,*,*,i] = $
         a_tmp[*,*,*,0] * !p.t[i,0] + $
         a_tmp[*,*,*,1] * !p.t[i,1] + $
         a_tmp[*,*,*,2] * !p.t[i,2]
        for i= 0,2 do a_out[*,*,*,i] = turn_3d(a_out[*,*,*,i], $

x_angle,0.,0.,_extra=_extra, $

resize=resize,verbose=verbose)
      endif
      if y_angle ne 0. then begin
        a_tmp =  a_out
        t3d,/reset,rotate=[0.,y_angle,0.]
        for i= 0,2 do a_out[*,*,*,i] = $
         a_tmp[*,*,*,0] * !p.t[i,0] + $
         a_tmp[*,*,*,1] * !p.t[i,1] + $
         a_tmp[*,*,*,2] * !p.t[i,2]
        for i= 0,2 do a_out[*,*,*,i] = turn_3d(a_out[*,*,*,i], $

0.,y_angle,0.,_extra=_extra, $

resize=resize,verbose=verbose)
      endif
      return,a_out

    endif                       ; case a is a vector (4D)


      if keyword_set(verbose) then print,' turn_3d: simple case'
      a =  reform(a_in)
      if (size(a))(0) ne 3 then stop,' A must be 3D'

      x_size = (size(a))(1)
      y_size = (size(a))(2)
      z_size = (size(a))(3)

      max_size = x_size > y_size > z_size
```

```
; let's do a 50% larger 3D array containing the input 3D array at his
"center"

      new_size = fix(max_size*1.5) + 1
      b = congrid(a*0.,new_size,new_size,new_size)

      b[(new_size-x_size)/2:(new_size-x_size)/2+x_size-1,$
        (new_size-y_size)/2:(new_size-y_size)/2+y_size-1,$
        (new_size-z_size)/2:(new_size-z_size)/2+z_size-1] = a

; Z-rotation
      if z_angle ne 0. then begin
         for z = 0,new_size-1 do b[*,*,z] =
rot(reform(b[*,*,z]),z_angle,$
                                      _extra=_extra)
      endif

; X-rotation
      if x_angle ne 0. then begin
         for x = 0,new_size-1 do b[x,*,*] =
rot(reform(b[x,*,*]),x_angle,$
                                      _extra=_extra)
      endif

; Y-rotation
      if y_angle ne 0. then begin
         for y = 0,new_size-1 do b[*,y,*] =
rot(reform(b[*,y,*]),-y_angle,$
                                      _extra=_extra)
      endif

      if keyword_set(resize) then b = $
       b[(new_size-x_size)/2:(new_size-x_size)/2+x_size-1,$
         (new_size-y_size)/2:(new_size-y_size)/2+y_size-1,$
         (new_size-z_size)/2:(new_size-z_size)/2+z_size-1] $
      else if keyword_set(verbose) then $
       print,' Size changed by: ',float(new_size) / float(max_size)

      if keyword_set(conserv) then b = b / total(b) * total(a)

      return,b

end
```

Subject: Re: arbitrary rotation of 3-d arrays
Posted by steinhh on Fri, 11 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

> has anybody out there in idl-land written or seen code to apply arbitrary
> rotations to 3-d arrays???

Although others have posted solutions using the t3d style
rotations, you might want to look at the procedures below.
I don't really like the idea of using a (global) system
variable designed for 3d *graphics* as "the" temporary
variable to accumulate all kinds of 3d manipulations...

I'm sorry for the complete lack of documentation ... this
was something I did to experiment myself towards an understanding
of such rotations.. ROT3DMATRIX returns an array that to
be applied like this

```
  ROTATED_XYZ = ROT3DMATRIX([alphax,alphay,alphaz]) ## XYZ_ARRAY
```

(Note that [alphax,alphay,alphaz] is wrt. a fixed coordinate
system, the axes don't move after each partial rotation...
this may be different from the way t3d applies its input..)

Also, I wrote a test application (ROTATE_SCREW) that uses
direct graphics to manipulate 3D objects on screen with the
help of a trackball object.

Try clicking either left *and* middle buttons to rotate the
box, and the (two!) corkscrews inside the box. Double-clicking
the middle button changes the "sense" of how the "loose"
corkscrew is rotated - wrt the *box* coordinate system or wrt
the *screen* (sort of) coordinate system.... You may get the
difference if you e.g. turn the box 180 degrees around and
try manipulating the loose corkscrew...

Ok, here goes,

Stein Vidar

```
;;
;; Return rotation matrix such that
```

```
;; rot3dmatrix([alphax,alphay,alphaz]) ## [X,Y,Z] gives your [X,Y,Z]
;; rotated alphax radians about the x axis, then alphay radians about
;; the y axis, and finally alphaz radians about the z axis. Note that
;; the axes are kept fixed. This allows an inverse operation to to
;; return alphax, alphay, alphaz from a given rotation matrix.

FUNCTION rot3dmatrix_angles,m

  m = double(m)

;  From mathematica:
;
;  mm = [[Cy Cz, Cz Sx Sy - Cx Sz, Cx Cz Sy + Sx Sz   ], $
;        [Cy Sz, Cx Cz + Sx Sy Sz, -(Cz Sx) + Cx Sy Sz],$
;        [-Sy  , Sx Cy           ,     Cx Cy         ]]


  cosyzero = (m(1,2) EQ 0 AND m(2,2) EQ 0)

  IF NOT cosyzero THEN BEGIN

     cys = 1

     xfind = atan(cys*m(1,2),cys*m(2,2)) ;; May be wrong quadrants!
     zfind = atan(cys*m(0,1),cys*m(0,0))

     sinx = sin(xfind) & sxgood = (abs(sinx) GT 0.05)
     cosx = cos(xfind) & cxgood = (abs(cosx) GT 0.05)

     wt = double(sxgood+cxgood)
     cosy = ((sxgood ? m(1,2)/sinx : 0)+(cxgood ? m(2,2)/cosx : 0))/wt

     yfind = atan(-m(0,2),cosy)

     return,[xfind,yfind,zfind]
  END

  ;; Cos[y] == 0 => yfind = +/- Pi/2

  yfind = atan(-m(0,2),0)

  ;; From mathematica we find that

  ;; MatrixForm[TrigFactor[r[x,+/-Pi/2,z]]]
  ;;
  ;; = [[ 0,  +/-Sin[x-z], +/-Cos[x+z]  ],$
  ;;    [ 0,     Cos[x-z],    -Sin[x-z] ],$
  ;;    [ -/+1,         0,           0  ]]
```

```
  ;; We thus arbitrarily set z = 0 and get:

  zfind = 0
  xfind = atan(-m(2,1),m(1,1))

  return,[xfind,yfind,zfind]
END


FUNCTION rot3dmatrix,alpha,inverse=inverse

  IF keyword_set(inverse) THEN return,rot3dmatrix_angles(alpha)

  alpha = double(alpha)
  ca = cos(alpha)
  sa = sin(alpha)

  mx = [[     1,     0,      0],$
        [     0, ca(0), -sa(0)],$
        [     0, sa(0),  ca(0)]]

  my = [[ ca(1),     0,  sa(1)],$
        [     0,     1,      0],$
        [-sa(1),     0,  ca(1)]]

  mz = [[ ca(2),-sa(2),      0],$
        [ sa(2), ca(2),      0],$
        [     0,     0,      1]]

  return,mz ## (my ## mx)
END

 ;------------------------------------------------------------ -


PRO plotcube,xr,yr,zr

  xi = [0,1,1,0,0]
  yi = [0,0,1,1,0]
  zi = [0,0,0,0,0]

  plots,xr(xi),yr(yi),zr(zi),/t3d,/data
  plots,xr(xi),yr(yi),zr(zi+1),/t3d,/data
  FOR i=0,4 DO plots,xr(xi([i,i])),yr(yi([i,i])),zr([0,1]),/t3d,/data

END
```

```
PRO rotate_screw,rotation

  ;; Create widget draw window

  xs = (ys=512) ;; Size of draw window

  id = widget_base()
  dummy = widget_draw(id,xsize=xs,ysize=ys,/button_ev,/motion)
  widget_control,id,/realize
  widget_control,dummy,get_value=win
  wset,win

  xrange = [(xmin=-10), (xmax=10)]
  yrange = [(ymin=-10), (ymax=10)]
  zrange = [(zmin=-10), (zmax=10)]

  !x.s = [-xmin,1.0]/(xmax-xmin)
  !y.s = [-ymin,1.0]/(ymax-ymin)
  !z.s = [-zmin,1.0]/(zmax-zmin)

  theta = findgen(120)/199.0*2*!PI
  x = cos(20*theta)
  y = sin(20*theta)
  z = 5*theta

  xyz = [[x],[y],[z]]
  t3d,/reset,translate=-[.5,.5,.5] & xyzform = !P.t

  ;; X/Y/Z "axis" vectors for easy drawing

  xa = 8*[[0,1],[0,0],[0,0]]
  ya = 8*[[0,0],[0,1],[0,0]]
  za = 8*[[0,0],[0,0],[0,1]]

  h = [.5,.5,.5]
  persp = 5
  scale = .6

  ;; Build the initial viewing matrix

  t3d,/reset,trans=-h
  t3d,scale=.6*[1,1,1]
  IF n_elements(rotation) EQ 3 THEN BEGIN
     t3d,rotate=rotation*!radeg
     print,"Rotation"
  END
```

```
t3d,trans=h

track = obj_new('trackball',[xs/2.,ys/2.0],0.25*xs)
track2 = obj_new('trackball',[xs/2.,ys/2.0],0.25*xs)

inspace = 1

REPEAT BEGIN
  t = !P.t
  t3d,trans=-h
  angles = rot3dmatrix(!p.t(0:2,0:2),/inverse)
  t3d,perspect=persp
  t3d,trans=h

  erase

  plotcube,xrange,yrange,zrange

  plots,transpose(xa),/t3d,/data
  plots,transpose(ya),/t3d,/data,color=140
  plots,transpose(za),/t3d,/data,color=100

  xyouts,xa(1,0),xa(1,1),z=xa(1,2),"X",/t3d,/data
  xyouts,ya(1,0),ya(1,1),z=ya(1,2),"Y",/t3d,/data
  xyouts,za(1,0),za(1,1),z=za(1,2),"Z",/t3d,/data

  plots,x,y,z,/t3d,/data
  plots,transpose(xyz),/t3d,/data

  xyouts,.1,.15,string(angles(0))+"!c"+string(angles(1))+$
    "!c"+string(angles(2))+"!c"+"insp:"+string(inspace),/normal

  !P.t = t
  empty
  ev = widget_event(id)
  IF ev.press EQ 2 AND ev.clicks EQ 2 THEN inspace = (inspace + 1) MOD 3

  xformq = track->update(ev,transform=xform,mouse=1b)
  IF xformq THEN BEGIN
    t3d,translate=-h
    !P.t = xform ## !p.t
    t3d,translate=h
  END

  xformq = track2->update(ev,transform=xform,mouse=2b)
  IF xformq THEN BEGIN
    IF inspace EQ 1 THEN BEGIN
```

```
      ;; Now - rotate/shift the screw into the orientation it has on the
      ;; screen, then apply trackball rotation, then rotate/shift back
      ;; into it's own space.

      xform = invert(!p.t) ## xform ## !p.t
    END ELSE IF inspace EQ 2 THEN BEGIN
      ;; Rotate the screw "in its own data space".
      ;; Apply inverse transform on xyz,
      ;;
      xform = xyzform ## xform ## invert(xyzform)
    END

    ;; Apply the resulting transform on the screw. The xform really
    ;; includes a shift (since we placed it centered on the screen, not on
    ;; the coordinate axes), but this is not taken into account since
    ;; we're using only (0:2,0:2) of the resulting transform

    xyz = xform(0:2,0:2) ## xyz

    ;; We need to keep track of the transform that has been applied
    ;; in order to do transformations in
    xyzform = xform ## xyzform
  END
 END UNTIL ev.press EQ 4

 widget_control,id

 rotation = angles
END
```

Subject: Re: arbitrary rotation of 3-d arrays
Posted by Michael Asten on Fri, 11 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Not sure how sophisticated you want to be here.
The !P.T structure makes it easy to start in idl. See online documentation
under "Three-dimensional graphics" for starters - including the demo routine
HOUSE.PRO
The demo shows how to rotate coordinates for the purpose of making a 2D
projection, but the same tools work for rotating 3D arrays as abstract
entities.

When I want to rotate a set of coordinates given by vectors Xin, Yin, Zin,
in 3D, I generate a !P.T transformation using the following
; 3D coordinates before rotation are in vectors Xex,Yex,Zex
t3d,/reset
 t3d,rotate=[0.,0.,plu] & t3d,rotate=[-dip,0.,0.] & t3d,rotate=[0.,0.,str]

```
 t3d,translate=[xsh,ysh,zsh]
;  we have set up t3d to rotate a body thru a strike(-azimuthal)  angle str,
;     a dip angle dip, and a plunge angle plu,
; and we have added a translation of position of the refernce point of the
body
;   to (xsh,ysh,zsh).
;
;  we now execute the rotation and translation
do_rotation,xex,yex,zex,xrot,yrot,zrot
;  and can plot or otherwise operate on the new rotated coordinates howsoever
we ;  please
end  ;  of demo
```

The routine to do the rotation and translation is simply:

```
;  routine to perform rotation of n points in x[0:n-1],y[  ] and z[  ]
;     using the existing !P.T transformation
;  input:  xin,yin zin being arrays of reals
;  output: xout,yout,zout  being arrays of reals,  for transformed points
;  Author: Michael Asten, Monash University, Melbourne Australia.  June 1999.

 pro do_rotation,xin,yin,zin,xout,yout,zout
   P=fltarr(4,n_elements(xin))
   P[0,*]=xin & P[1,*]=yin & P[2,*]=zin & P[3,*]=1.
   P=transpose(P)
   Prot=P#!P.T  ;  do rotation and shift
   Prot=transpose(Prot)
   xout=Prot[0,*]/Prot[3,*]
   yout=Prot[1,*]/Prot[3,*]
   zout=Prot[2,*]/Prot[3,*]
 end
```

"D. Mattes" wrote:

```
>  hello idl users:
>  has anybody out there in idl-land written or seen code to apply arbitrary
>  rotations to 3-d arrays???
>
>  thanks in advance!
>
>  david mattes
```

Subject: Re: arbitrary rotation of 3-d arrays
Posted by morisset on Sat, 12 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Foster wrote:

> The previous posts discussing the use of T3D were all assuming that
> once the transformation of coordinates was performed, one would have
> to then use interpolation to actually transform the data. At least in
> my post I simply forgot to mention this important step. (oops!)

And perhaps the original question was 'How do interpolate in a 3D
cube?' !!

> My main concern with your method is that it is using 2D techniques
> to perform a 3D task, and I believe will invariably suffer from poorer
> performance. Both methods will require large amounts of memory, but
> in your method you make entire copies of the original data-set, and
> three times! Often the data for such an operation will be huge, as
> in the case of medical images. The performance penalties this copying
> will incur, as well as the use of for loops to process each set of
> 2D images through the data, will make this method much slower than
> the use of T3D and interpolation.

Once more, the point is the interpolation, not to get the coordinate
matrix. I'm not sure that a 3D interpolation will be faster than N
2D interpolation. Since the poly_2d function used in the rsi ROT
function is not available, it's not possible to 'have a look and
generalize'!

> I would also argue that it would be less accurate, since you are
> performing interpolation three times in succession basically, once
> within each 2D plane, whereas the T3D method will transform the
> coordinates mathematically and then interpolate the original data
> once from those computed coordinates. Quite a different animal.

I agree with you and at the time I was needing this turn_3d, I tried
to make the 3D interpolation after doing the t3d transformation.
As you see, I didn't succeed (well, I didn't tried a lot of time,
'cause my datas are 'just' 100^3)! And the use of 2D slide by slide
was better, 'cause I finally just had to make one rotation ;-)

Anyway, the question remains: where is a 3D interpolation function???

Best regards.
Chris.

Subject: Re: arbitrary rotation of 3-d arrays
Posted by D. Mattes on Tue, 15 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

IDL users: thanks for the lively discussion regarding this topic. i never did find a canned procedure for me to use, but i implemented a 3D volume rotation class method using VERT_T3D and INTERPOLATE. i submit it here for anybody who would like it, and for suggestions for optimization and improvement.

cheers,
david mattes

```
------------------CUT HERE------------------------------
pro ImageClass::ApplyRigidTransformation,U,t
    ;U is a 3x3 unitary rotation matrix
    ;t is a 3x1 vector of x,y,z translations

    ;self is the local object reference with members:
    ;   self.xdim, self.ydim, self.zdim  (size of volume)
    ;   self.volume  (pointer to 3D volume data)

    ;build 4x4 homogeneous transformation matrix
   localT=fltarr(4,4)
   localT(0:2,0:2)=U
   localT(3,0:2)=t
   localT(3,3)=1.

    ;build x,y,z interpolation points
   vert_size=LONG(self.xdim)*$
      LONG(self.ydim)*$
      LONG(self.zdim)
   verts=fltarr(3,vert_size)
   count=0L
   for i=0,self.zdim-1 do begin
      for j=0,self.ydim-1 do begin
         for k=0,self.xdim-1 do begin
            ;notice index swapping here!!!!!!!!!!!!
            ;for our image, the x coordinate is the most
            ;quickly varying, so it must also be this way for the
            ;interpolate locations
           verts(*,count)=[k,j,i]
           count=count+1
         endfor
      endfor
   endfor

    ;verts are the location points for which we want
    ;interpolates...really just the indices in the volume array.
```

```
    ;we transform these indices using the localT transformation
    ;matrix, and pass them to the interpolate function.
    ;notice: verts is 3 x (n*m*l) 2D matrix for a n x m x l array!!!
   verts=VERT_T3D(verts,MATRIX=localT,/NO_COPY,/NO_DIVIDE)

    ;right now, use missing=-1 for values outside input data range.
    ;cubic interpolation is not supported for 3-d case
   *self.volume=INTERPOLATE(TEMPORARY(*self.volume),$
      verts(0,*),verts(1,*),verts(2,*),$
      MISSING=-1)

    ;the interpolate function returns a 1-d array of interpolated
    ;points, which must be resized into the original array shape.
   *self.volume=REFORM(*self.volume,$
      self.xdim,$
      self.ydim,$
      self.zdim,/OVERWRITE)

end
```