```fortran
module ConicalRWE3D_wemig
   use sep            !! basic SEPlib routine I/O module
   use util_          !! handle axes module
   use ConicalKernel3D!! GOGI's 80 degree code
  use sep_timers_mod !! Timing
  use fft_mod       !! Set up FFTW
  implicit none
  integer, private :: ne1,ne2,ne3,nw,nxtap,nytap,nx1,nx2,nx3,nt
  integer, private :: ns1,ns2,nr1,nr2,method,nyint,nxint,wix,wiy
  real, private :: de1,de2,de3,dw,oe1,oe2,oe3,ow,dr1,dr2,or1,or2
  real, private :: pi,r0,eps,ox1,ox2,ox3,dx1,dx2,dx3,os1,os2,ot,dt
  real, private :: ds1,ds2,minang,maxang
  real, allocatable,private::ww(:),tap(:,:),Timg(:,:,:,:),vmin(:),tline(:),rline(:)
  complex, allocatable, private :: rax(:,:,:),sax(:,:,:)
  logical, private :: verbose,source_norm,timing
contains
  !------------------------------------------------------------------------
  !! . . Initialization routine - parameter passing and array allocation
  subroutine wemig_axes_init(aE1,aE2,aE3,aW,aX1,aX2,aX3,aS1,aS2,aR1,aR2,aT)
    type(myaxis)        :: aE1,aE2,aE3,aW,aX1,aX2,aX3,aS1,aS2,aR1,aR2,aT

    !! . . Cartesian
    nx1=ax1%n; nx2=ax2%n; nx3=ax3%n
    ox1=ax1%o; ox2=ax2%o; ox3=ax3%o
    dx1=ax1%d; dx2=ax2%d; dx3=ax3%d

    !! . . RWE
    ne1=aE1%n; ne2=aE2%n; ne3=aE3%n;
    de1=aE1%d; de2=aE2%d; de3=aE3%d;
    oe1=aE1%o; oe2=aE2%o; oe3=aE3%o;

    !! . . Input data
    nr1=aR1%n; nr2=aR2%n;
    dr1=aR1%d; dr2=aR2%d;
    or1=aR1%o; or2=aR2%o

    !! . . Shot axis
    ns1=as1%n; ns2=as2%n;
    os1=as1%o; os2=as2%o;
    ds1=as1%d; ds2=as2%d

    !! . . Frequency
    pi=acos(-1.)
    nw=aW%n; dw=aW%d*2.*pi; ow=aW%o*2.*pi

    !! . . Inline coordinate system tilt
    nt = aT%n; ot = aT%o;   dt = aT%d

  end subroutine wemig_axes_init
  !------------------------------------------------------------------------
  !! . . Initialization routine - parameter passing and array allocation
  subroutine wemig_param_init(verbose_in,r0_in,minang_in,maxang_in,eps_in,method_in)
    integer :: method_in,i1,ie3,ie1,ie2
    real    :: r0_in,rbeg_in,rend_in,eps_in,minang_in,maxang_in,d2,o2
    logical :: verbose_in

    !! For OMP
    integer, external :: omp_get_num_threads,omp_get_thread_num
    integer  :: nth,ith

    pi=acos(-1.)
    eps=eps_in
    method = method_in
    verbose=verbose_in;
    r0=r0_in
    minang = minang_in
    maxang = maxang_in
    call from_param("nxtap" ,nxtap, 40) !! Width of cosine taper
    call from_param("nytap" ,nytap, 40) !! Width of cosine taper
    call from_param("timing",timing,.false.)
    call from_param("wix",wix,3) !! Wavefield injection parameter - x direction
    call from_param("wiy",wiy,3) !! Wavefield injection parameter - y direction
    !$OMP PARALLEL
    nth = omp_get_num_threads()
    !$OMP END PARALLEL

    !! Velocity and fields
    allocate( vmin(ne3),Timg(ne1,ne2,ne3,nth) )
    allocate( tap(ne1,ne2),sax(ne1,ne2,nth),rax(ne1,ne2,nth),ww(nth))

    !! Initalize
    !$OMP PARALLEL DO PRIVATE(ith,ie3,ie2,ie1)
    do ith=1,nth
      do ie3=1,ne3
        do ie2=1,ne2
          do ie1=1,ne1
            Timg(ie1,ie2,ie3,ith)=0.
          end do
        end do
      end do
    end do
    !$OMP END PARALLEL DO
    call rwetaper_init() !! Initialize the taper program
```

```fortran
    allocate( tline(ne3),rline(ne2) )
    call GET_LINES(tline,rline)

  end subroutine wemig_param_init
  !-----------------------------------------
  !! . . Get x- and z- initial shifts
  subroutine COORD_SHIFTS(method,tangle,xinit,zinit)
    integer :: method
    real :: tangle,xinit,zinit

    !! Z-shift
    if (method .eq. 1) then
      zinit=0.
    else
      zinit = abs((dble(nr1-1.)*dr1)*sin(tangle) )
    end if

    !! . . x_init shift
    if (method .eq. 1) then
      xinit =0.
    else
      if (tangle .lt. 0.) then
!         xinit = -sin(tangle)*zinit
!         xinit = abs(r0*sinh(tline(ne3/2))*sin(tangle)/2.)+(nr1*dr1+or1)
         xinit =  -abs(r0*sinh(tline(ne3/2))*sin(tangle)/2.)
     else
         xinit = sin(tangle)*zinit/2.
!        xinit =  abs(r0*sinh(tline(ne3/2))*sin(tangle)/2.)
      end if
    end if
    write(0,*)'           * x_init,z_init,tangle',xinit,zinit,tangle

  end subroutine COORD_SHIFTS

  !------------------------------------------------------------------------
  !! . . Main wavefield extrapolation and imaging routine
  subroutine wemig(rwf,swf,RImg,ishot,iT,is2,RVel,fields)
    integer :: iw,ie1,ie2,ie3,inn,is,id,ip,ishot,iT,is2
    integer, allocatable :: done_inject(:)
    real    :: vtest,rarg,sarg,ss,w,z_init,tangle,x_init
    real    :: RImg(:,:,:),fields(:,:),RVel(:,:,:)
    real, allocatable :: minz(:)
    complex :: rwf(:,:,:),swf(:,:,:)

    !! For OMP
    integer, external :: omp_get_num_threads,omp_get_thread_num
    integer :: nth,ith
    logical :: dofilt

    !! Tilt angle to radians
    tangle = pi*( (iT-1.)*dT+oT) / 180.

    call COORD_SHIFTS(method,tangle,x_init,z_init)

    !$OMP PARALLEL
    nth = omp_get_num_threads()
    !$OMP END PARALLEL

    allocate ( done_inject(nth),minz(nth) )
    done_inject=0.
    sarg =-1.;    rarg = 1.

    write(0,*)'           * Working on shot ',ishot,' of ',ns1*ns2

    !! Get Minimum values for the Fourier filtering
    do ie3=1,ne3
      vtest=100000.*maxval(RVel(:,:,ie3))
      do ie2=1,ne2
        vtest = min(vtest, minval( RVel(:,ie2,ie3) / fields(ie2,ie3) ) )
      end do
      vmin(ie3) = vtest
    end do

    if (method .eq. 1) then
      call shot_ker_init(nth, ne1, ne2, de1,    de2,   de3, 0., 0., .11)
    else
      call shot_ker_init(nth, ne1, ne2, de1, r0*de2,r0*de3, 0., 0., .11)
    end if

    !! . . Initialize FFT2D algorithm
    call fftw_init(sax,rax)

    !$OMP PARALLEL DO SCHEDULE(DYNAMIC) PRIVATE(iw,id,ie3,ie2,ie1) &
    SHARED(nw,ow,dw,ne3,ne2,ne1,rarg,sarg,Timg,sax,rax,fields,tap,done_inject,minz)

    Frequency_loop: do iw=1,nw  !! FREQUENCY LOOP
      id = omp_get_thread_num()+1
      done_inject (id) = 0
      ww(id) = dble(iw-1.)*dw+ow

      !! SAX and RAX  propagation wavefields, SWF and RWF  data wavefields
      do ie2=1,ne2
```

```
      do ie1=1,ne1
        sax(ie1,ie2,id) = 0.
      end do
    end do
    do ie2=1,ne2
      do ie1=1,ne1
        rax(ie1,ie2,id) = 0.
      end do
    end do

!      !! . . Define the Source wavefield
!      do ie2=1,ne2
!        do ie1=1,ne1
!          sax(ie1,ie2,id) = swf(ie1,ie2,iw)
!        end do
!      end do
!      !! . . Define Receiver wavefield
!      do ie2=1,ne2
!        do ie1=1,ne1
!          rax(ie1,ie2,id) = rwf(ie1,ie2,iw)
!        end do
!      end do

    EXTRAP_LOOP: do ie3=2,ne3

      minz(id) = abs(r0)*sinh(tline(ie3))*cos(tangle)-&
      abs( (dble(nE1-1.)*dr1) * sin(tangle) ) - z_init

      !! Cycle if shallowest ray depth is greater than max image
      if ( method .ne. 1 .and. minz(id) .gt. (nx3-1.)*dx3+ox3 ) cycle

      !! Propagate WFs in depth
      if ( done_inject(id) .eq. 0 ) then
        call  INJECT_WAVEFIELD (id,iw,ie3-1,swf,sax,x_init,z_init,tangle,it,is2)
        call  INJECT_WAVEFIELD (id,iw,ie3-1,rwf,rax,x_init,z_init,tangle,it,is2)
        call  INJECT_TEST (id,ie3,minz(id),done_inject,tangle)
      end if

      !! . . Taper
      call rwetaper(sax,id)
      call rwetaper(rax,id)

      !! . . Step Source and Receiver wavefields
      call shot_ker_onestep(id,sax,(Rvel(:,:,ie3)+Rvel(:,:,ie3-1))/2.,&
      (fields(:,ie3)+fields(:,ie3-1))/2.,ww(id),sarg)
      call shot_ker_onestep(id,rax,(Rvel(:,:,ie3)+Rvel(:,:,ie3-1))/2.,&
      (fields(:,ie3)+fields(:,ie3-1))/2.,ww(id),rarg)

      !! . . Taper
      call rwetaper(sax,id)
      call rwetaper(rax,id)

      !! . . High-angle filter  FFT to kx
      call SAX_Forward(sax,id)
      call RAX_Forward(rax,id)

      !! . . Phase correction routine
      call wemig_phs_correction(id,sax,ww(id),vmin(ie3),sarg,&
      (fields(:,ie3)+fields(:,ie3-1))/2.)
      call wemig_phs_correction(id,rax,ww(id),vmin(ie3),rarg,&
      (fields(:,ie3)+fields(:,ie3-1))/2.)

      !! . . FFT to x
      call SAX_Backward(sax,id)
      call RAX_Backward(rax,id)

      !! . . Taper
      call rwetaper(sax,id)
      call rwetaper(rax,id)

      !! . . Imaging condition
      do ie2=1,ne2
        do ie1=1,ne1
          Timg(ie1,ie2,ie3,id)=Timg(ie1,ie2,ie3,id)+real(rax(ie1,ie2,id)*conjg(sax(ie1,ie2,id)) )
        end do
      end do

    end do EXTRAP_LOOP

    done_inject(id) = 0.
    if (verbose .and. id .eq. 1 ) &
    write(0,'(a,i5,i5,i5,2F8.2,2F11.4)') &
    '          * ST: F%,is,ns,w,im: ',floor(100*dble(iw)/dble(nw)),ishot,ns1*ns2,&
    ww(id)/2./pi,100.*maxval(Timg(:,:,:,id))

  end do Frequency_loop
  !$OMP END PARALLEL DO

  !! SUMMATION OVER OUTSIDE IMAGE CONDITION
  do ith=1,nth
    !$OMP PARALLEL DO PRIVATE(ie3,ie2,ie1)
    do ie3=1,ne3
      do ie2=1,ne2
```

```
        do ie1=1,ne1
          RImg(ie1,ie2,ie3)=RImg(ie1,ie2,ie3)+Timg(ie1,ie2,ie3,ith)
        end do
      end do
    end do
    !$OMP END PARALLEL DO
  end do
  write(0,*) '          * IMAGE maximum is: ',maxval(RImg)

  !$OMP PARALLEL DO PRIVATE(ie3,ie2,ie1,ith)
  do ith=1,nth
    do ie3=1,ne3
      do ie2=1,ne2
        do ie1=1,ne1
          Timg(ie1,ie2,ie3,ith) = 0.
        end do
      end do
    end do
  end do
  !$OMP END PARALLEL DO

  deallocate (done_inject , minz )
  call shot_ker_release()

end subroutine wemig
!-------------------------------------------------------------------
!! . . Inject the wavefields into the global wavefields
subroutine INJECT_WAVEFIELD(id,iw,ie3,xwf,xax,x_init,z_init,tangle,it,is2)
  integer :: id,ie3,iE1,iE2,ix,iy,iz,iw,jx,jy,is2,it
  real :: x,y,z,xx,yy,zz,ddx,ddy,ddz,r,dr,s,bp,x_init,z_init
  real :: tangle,x0,y0,z0
  complex :: xwf(:,:,:),xax(:,:,:)

  if ( method .eq. 1) then !! If Cartesian
    z = (ie3-1.)*de3+oe3
    ddz=0.
    if (ne2 .eq. 1) then !! If 2D

      dr = sqrt(dx1**2 + dx3**2)

      do iE1=1,ne1

        x = (ie1-1.)*dr1+or1
        ix=floor( ( x - or1 ) / dr1 ) + 1
        if (ix .lt. 1 .or. ix .gt. nr1 ) cycle

        do jx=ix-wix, ix+wix
          if ( jx .lt. 1 .or. jx .gt. nr1) cycle
          xx = or1 + (jx-1.)*dr1
          ddx =  xx-x
          r = sqrt( ddx**2 +  ddz**2) / dr
          s = min( sin(r) / r , 1.0 )
          xax(iE1,1,id) = xax(iE1,1,id) + s*xwf(jx,1,iw)
        end do
      end do

    else !! If 3D
      dr = sqrt( dx1**2 + dx2**2 + dx3**2)
      ddz=0.
      do iE2=1,ne2
        y = dble(ie2-1.)*dr2+or2
        iy=floor( ( y - or2 ) / dr2 ) + 1.5
        if (iy .lt. 1 .or. iy .gt. nr2 ) cycle

        do iE1=1,ne1
          x = dble(ie1-1.)*dr1+or1
          ix=floor( ( x - or1 ) / dr1 ) + 1.5
          if (ix .lt. 1 .or. ix .gt. nr1 ) cycle

          do jx=ix-wix, ix+wix
            if ( jx .lt. 1 .or. jx .gt. nr1) cycle
            xx=or1 + (jx-1.)*dr1
            ddx=xx-x

            do jy=iy-wiy,iy+wiy
              if ( jy .lt. 1 .or. jy .gt. nr2  ) cycle
              yy=or2 + (jy-1.)*dr2
              ddy=yy-y

              r = sqrt( ddx**2 + ddy**2 + ddz**2) / dr
              s = min( sin(r) / r , 1.0 )
              xax(iE1,iE2,id) = xax(iE1,iE2,id) + s*xwf(jx,jy,iw)

            end do
          end do
        end do
      end do
    end if !! End If 2D/3D

  else !! Non-Cartesian

    if (ne2 .eq. 1 ) then !! If 2D
      dr = sqrt(dx1**2 + dx3**2)
```

```
      x0 = abs(r0)*sinh(tline(ie3))*sin(tangle)
      z0 = abs(r0)*sinh(tline(ie3))*cos(tangle)

      do iE1=1,ne1
        if (tangle .gt. 0.) then
          z = z0 - (    iE1-1.)*dr1*sin(tangle)
        else
          z = z0 + (ne1-iE1+1.)*dr1*sin(tangle)
        end if
        iz=floor( abs( z / (wix*dx3) )) + 1.5
        ddz = z

        if (iz .eq. 1) then

          x=or1+(iE1-1.)*dr1*cos(tangle)+x0-x_init

          ix= ( x - or1 ) / dr1 + 1.5

          if (ix .lt. 1 .or. ix .gt. nr1 ) cycle

          do jx=ix-wix, ix+wix
            if ( jx .lt. 1 .or. jx .gt. nr1) cycle
            xx = or1 + (jx-1.)*dr1
            ddx = xx-x
            r = sqrt( ddx**2 +  ddz**2) / dr
            s = min( sin(r) / r , 1.0 )
            xax(iE1,1,id) = xax(iE1,1,id) + s*xwf(jx,1,iw)
          end do
        end if
      end do
    else !! If 3D
      dr = sqrt( dx1**2 + dx2**2 + dx3**2)
      y0 = os2+(is2-1.)*ds2
      E1Loop: do iE1=1,ne1
        E2Loop: do iE2=1,ne2

          !! . . Locate the z-position index
          if (tangle .gt. 0.) then
            z = abs(r0)*sinh(tline(ie3))*sin(rline(iE2))*&
            cos(tangle)-(    iE1-1.)*dr1*sin(tangle)
          else
            z = abs(r0)*sinh(tline(ie3))*sin(rline(iE2))*&
            cos(tangle)+(ne1-iE1+1.)*dr1*sin(tangle)
          end if

          iz=floor( abs( z / (wix*dx3) )) + 1.5
          ddz = z
          if (iz .eq. 1) then

            !! X position
            x  = or1 + (ie1-1.)*dr1*cos(tangle) +&
            abs(r0)*sinh(tline(ie3))*sin(rline(iE2))*sin(tangle) - x_init
            ix = ( x - or1 ) / dr1 + 1.5
            if (ix .lt. 1 .or. ix .gt. nr1 ) cycle

            !! Y position
            y  = y0 - abs(r0)*cosh(tline(ie3))*cos(rline(iE2))


            iy = floor( (y - or2 - y0 )/ dr2 ) + 1.5


            if (iy .lt. 1 .or. iy .gt. nr2 ) cycle

            X1Loop: do jx=ix-wix, ix+wix
              if ( jx .lt. 1 .or. jx .gt. nr1 ) cycle
              xx=or1 + (jx-1.)*dr1
              ddx=xx-x

              X2Loop: do jy=iy-wiy,iy+wiy
                if ( jy .lt. 1 .or. jy .gt. nr2 ) cycle
                yy=or2+(jy-1.)*dr2 + y0
                ddy=yy-y

                r = sqrt( ddx**2 + ddy**2 + ddz**2) / dr
                s = min( sin(r) / r , 1.0 )
                xax(iE1,iE2,id) = xax(iE1,iE2,id) + s*xwf(jx,jy,iw)

              end do X2Loop
            end do X1Loop
          end if
        end do E2Loop
      end do E1Loop

    end if
  end if !! End Cartesian/Non-Cartesian

end subroutine INJECT_WAVEFIELD

!-------------------------------------------------------------------
!! . . Inject_Test(id,ie3,rays,done_inject)
!! . . Test to see if we are beyond injection locations
!! . . done_inject=1 --> done injecting
!! . . done_inject=0 --> more injection needed
subroutine INJECT_TEST(id,ie3,zmin,done_inject,tangle)
  integer :: id,ie3,done_inject(:)
  real    :: zmin,tangle
  if (zmin  .gt. 3*dx3  ) done_inject(id) = 1
  if (tangle.eq. 0.     ) done_inject(id) = 1
end subroutine INJECT_TEST

!-------------------------------------------------------------------
!! . . Figure out which subset of plane waves correspond to a given tilt angle
subroutine PLANEWAVES_TODO(it,pwmin,pwmax)
  integer :: it,pwmin,pwmax,npw
  !! . . Find the start and end of the plane waves to do
  npw = ceiling(dble(ns1)/dble(nt))

  if (it .lt.  (nt+1)/2 )  then
     pwmin= 1+(it-1)*npw
     pwmax= it*npw
  else if (it .gt. (nt+1)/2 ) then
     pwmin= ns1 - (nt-it+1)*npw+1
     pwmax= ns1 - (nt-it  )*npw
  else
     pwmin=1+(it-1)*npw
     pwmax= ns1 - (nt-it  )*npw
  end if

end subroutine PLANEWAVES_TODO

!-------------------------------------------------------------------
!! . . Get tline and rline
subroutine GET_LINES(e3line,e2line)
  integer :: ie3,ie2
  real :: e2line(:),e3line(:)
  real :: d2,o2

  !! . . Define the extrapolation axis
  if (nr2 .ne. 1) then
     d2 = (maxang-minang)/180.*acos(-1.)/dble(nr2-1.)
     o2 = pi*minang/180.
  else
     d2 = (maxang-minang)/180.*acos(-1.)/dble(ne3-1.)
     o2 = acos(-1.)*minang/180.
  end if

  do ie3=1,ne3
     e3line(ie3)=(ie3-1.)*de3+oe3
  end do

  if (ne2 .eq. 1) then
     e2line(1) = pi/2.
  else
     do ie2=1,ne2
        e2line(ie2)=dble(ie2-1.)*d2+o2
     end do
  end if

end subroutine GET_LINES

!-------------------------------------------------------------------
!! . . Apply Taper to wavefield
subroutine rwetaper(dax,id)
  integer :: ie1,ie2,id
  complex :: dax(:,:,:)
  do ie2=1,ne2
     do ie1=1,ne1
        dax(ie1,ie2,id) = dax(ie1,ie2,id) * tap(ie1,ie2)
     end do
  end do
end subroutine rwetaper

!-------------------------------------------------------------------
!! . . Read in the parameters required for setting up geometry...etc
subroutine Read_In_Parameters(aE3,verbose,norm,nsx,nsy,nsz,eps,minang,&
maxang,method,newimage,nxint,nyint)
  type(myaxis) :: aE3
  integer:: norm,nsx,nsy,nsz,method,nxint,nyint
  real :: eps,minang,maxang,amax,b,c
  logical :: verbose,newimage
  call from_param("norm",norm,1)
  call from_param("nsx",nsx,2)
  call from_param("nsy",nsy,2)
  call from_param("nsz",nsz,2)
  call from_param("ne3",ne3,500)
  call from_param("oe3",oe3,0.)
  call from_param("de3",de3,0.003)
  call from_param("eps",eps,0.001)
  call from_param("minang",minang, 10.)
  call from_param("maxang",maxang,170.)
  call from_param("method",method,2)
  call from_param("verbose",verbose,.false.)
  call from_param("newimage",newimage,.false.)
  call from_param("nxint",nxint,3)
```

```fortran
  call from_param("nyint",nyint,3)
  aE3%n=ne3;    aE3%o=oe3;    aE3%d=de3
end subroutine Read_In_Parameters

!---------------------------------------------------------------------------
!! . . Read in the header information describing wavefield geometry
subroutine RWF_Info(aE1,aE2,aW,aS1,aS2,aR1,aR2)
  type(myaxis) ::    aE1,aE2,aW,aS1,aS2,aR1,aR2
  real :: o1,d1
  integer :: n1
  character(len=128) :: name
  name = 'rwf'
  call from_aux("rwf","n1",n1)
  call from_aux('rwf','o1',o1)
  call from_aux('rwf','d1',d1)
  call ainaxis(aE1,1,name)
  call ainaxis(aE2,2,name)
  call ainaxis(aW ,3,name)
  call ainaxis(aS1,4,name)
  call ainaxis(aS2,5,name)
  call ainaxis(aR1,1,name)
  call ainaxis(aR2,2,name)
end subroutine RWF_Info

!---------------------------------------------------------------------------
!! . . Read in the header information describing wavefield geometry
subroutine IMAGE_Out_Info(aX1,aX2,aX3)
  type(myaxis) ::          aX1,aX2,aX3
  call aouaxis(aX3,1,"image")
  call aouaxis(aX1,2,"image")
  call aouaxis(aX2,3,"image")
  call auxputch("esize","i",4,"image")
end subroutine IMAGE_Out_Info

!---------------------------------------------------------------------------
!! . . Read in the header information describing wavefield geometry
subroutine RIMAGE_Out_Info(aE1,aE2,aE3)
  type(myaxis) ::          aE1,aE2,aE3
  call aouaxis(aE1,1,"RImage.H")
  call aouaxis(aE2,2,"RImage.H")
  call aouaxis(aE3,3,"RImage.H")
  call auxputch("esize","i",4,"RImage.H")
end subroutine RIMAGE_Out_Info


!---------------------------------------------------------------------------
!! . . Read in the header information describing wavefield geometry
subroutine RVEL_Out_Info(aE1,aE2,aE3)
  type(myaxis) ::          aE1,aE2,aE3
  call aouaxis(aE1,1,"RVel.H")
  call aouaxis(aE2,2,"RVel.H")
  call aouaxis(aE3,3,"RVel.H")
  call auxputch("esize","i",4,"RVel.H")
end subroutine RVEL_Out_Info


!---------------------------------------------------------------------------
!! . . Read in the header information describing wavefield geometry
subroutine IMAGE_Info(aX1,aX2,aX3)
  type(myaxis) ::     aX1,aX2,aX3
  character(len=128) :: name
  name="image"
  call ainaxis(aX3,1,name)
  call ainaxis(aX1,2,name)
  call ainaxis(aX2,3,name)
end subroutine IMAGE_Info

!---------------------------------------------------------------------------
!! . . Read in the header information describing Cartesian velocity/image space
subroutine VEL_Info(aX1,aX2,aX3)
  type(myaxis) :: aX1,aX2,aX3
  character(len=128) :: name
  name='vel'
  call ainaxis(aX3,1,name)
  call ainaxis(aX1,2,name)
  call ainaxis(aX2,3,name)
end subroutine VEL_Info

!--------------------------------------------------------------------------
! . . Read in wavefields
subroutine READ_IN_WAVEFIELDS(name,xwf,nn1,nn2,nnw,n1beg,n1end,n2beg,n2end)
  integer            :: nn1,nn2,nnw,ii,n1beg,n1end,n2beg,n2end,nbytes
  complex            :: xwf(:,:,:)
  character(len=128)   :: name
  xwf=0.
  do ii=1,nnw
    call sreed(name,xwf(n1beg+1:nn1-n1end,n2beg+1:nn2-n2end,ii),8*(nn1-n1beg-n1end)*(nn2-n2beg-n2end) )
  end do
  write(0,*) '               * Max in:', maxval(abs(xwf))

end subroutine READ_IN_WAVEFIELDS
!---------------------------------------------------------------------------
!! . . Set up the Cartesian geometry for migration
subroutine CART_Setup(fields,gnorm)
  integer :: iE2,iE3
  real :: fields(:,:),gnorm(:,:)

  !$OMP PARALLEL DO PRIVATE(iE3,iE2)
  do iE3=1,ne3
    do iE2=1,ne2
      fields(iE2,iE3)= 1.
    end do
  end do
  !$OMP END PARALLEL DO

  !$OMP PARALLEL DO PRIVATE(iE3,iE2)
  do iE3=1,ne3
    do iE2=1,ne2
      gnorm(iE2,iE3)= 1.
    end do
  end do
  !$OMP END PARALLEL DO

  write(0,*) '           *************************************************'
  write(0,*) '           * '
  write(0,*) '           *       3D  Cartesian Migration'
  write(0,*) '           * '
  write(0,*) '           *             written by Jeff Shragge '
  write(0,*) '           *  '
  write(0,*) '           * CARTESIAN GRID CHECK '
  write(0,'(a,2f11.2,2f11.2)')'           * X-dir',or1,(nr1-1.)*dr1+or1
  write(0,'(a,2f11.2,2f11.2)')'           * Y-dir',or2,(nr2-1.)*dr2+or2
  write(0,'(a,2f11.2,2f11.2)')'           * Z-dir',oe3,(ne3-1.)*de3+oe3

end subroutine CART_Setup
!---------------------------------------------------------------------------
!! . .Set up the cylindrical elliptic  coordinate system for migration
subroutine CYNEL_Setup(fields,gnorm)
  integer  :: iE2,iE3
  real     :: fields(:,:),gnorm(:,:)

  !! . . Slowness stretch factor
  !$OMP PARALLEL DO PRIVATE(iE3,iE2)
  do iE3=1,ne3
    do iE2=1,ne2
      fields(iE2,iE3)= sqrt(sinh(tline(iE3))**2+sin(rline(iE2))**2)
   end do
  end do
  !$OMP END PARALLEL DO

  !! . . GNORM
  !$OMP PARALLEL DO PRIVATE(iE3,iE2)
  do iE3=1,ne3
    do iE2=1,ne2
      gnorm(iE2,iE3)= sqrt(sinh(tline(iE3))**2+sin(rline(iE2))**2)
    end do
  end do
  !$OMP END PARALLEL DO
  write(0,*) '           * ELLIPTIC GRID CHECK:'
  write(0,'(a,2f11.2,2f11.2)')'           * MIN/MAX Angles  :',minval(rline)*180./pi,maxval(rline)*180./pi
  write(0,'(a,2f11.2,2f11.2)')'           * MIN/MAX Ex step :',minval(tline),maxval(tline)
  write(0,'(a,2f11.2,2f11.2)')'           * Rline min/max   :',minval(rline)*180./pi,maxval(rline)*180./pi
end subroutine CYNEL_Setup

!---------------------------------------------------------------------------
!! . . Find the corresponding Cartesian velocity indicies
subroutine Get_Vel_Indicies(iix1,iix2,oo1,oo2,it,is2)
  integer :: iix1(:),iix2(:),it,is2
  real :: oo1(:),oo2(:),x_tmp,y_tmp,ee,x_init,z_init,tangle

  !! . . Tilt angle
  tangle = pi*((iT-1.)*dT+oT)/180.
write(0,*) tangle
  call COORD_SHIFTS(method,tangle,x_init,z_init)

  !! X1 index
  if (method .eq. 1) then
    oo1(1) = or1            + real(it-1.)*ds1
    oo1(2) = (nr1-1.)*dr1+or1 + real(it-1.)*ds1
  else
    oo1(1) = or1                 + real(it-1.)*ds1
    oo1(2) = or1 + dble(ne1-1.)*dr1 + real(it-1.)*ds1
  end if
  iix1(1) = floor   ((oo1(1) - ox1)/dx1) + 1
  iix1(2) = ceiling ((oo1(2) - ox1)/dx1)
  iix1(1) = max(1,min(iix1(1),nx1))
  iix1(2) = max(1,min(iix1(2),nx1))

  !! X2 index
  if (method .eq.1 ) then
    oo2(1) = os2 + (is2-1.)*ds2 + or2
    oo2(2) = os2 + (is2-1.)*ds2 + or2 + (nr2-1.)*dr2
  else
    if (ne2 .eq. 1) then
      oo2(1) = os2+(is2-1.)*ds2 + or2
      oo2(2) = os2+(is2-1.)*ds2 + or2
```

```
    else
      oo2(1) =−abs(r0)*cosh(tline(ne3))*cos(rline(1  ))+(is2−1.)*ds2+os2
      oo2(2) =−abs(r0)*cosh(tline(ne3))*cos(rline(ne2))+(is2−1.)*ds2+os2
    end if
  end if
  iix2(1) = floor   ((oo2(1) − ox2)/dx2) + 1
  iix2(2) = ceiling ((oo2(2) − ox2)/dx2)
  iix2(1) = max(1,min(iix2(1),nx2))
  iix2(2) = max(1,min(iix2(2),nx2))

end subroutine Get_Vel_Indicies
!−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
!! . . Find the locations of the image indicies
subroutine Get_Img_Indicies(extra,iix1,iix2,oo1,oo2,vox1,vox2,iT,is2,e1int1,e1int2)
  integer :: iix1(:),iix2(:),extra,iT,e1int1,e1int2,is2
  real :: oo1(:),oo2(:),vox1(:),vox2(:)
  real :: tangle,x_init,z_init

  !! Coordinate tilt angle
  tangle = pi*((iT−1.)*dT+oT)/180.

  call COORD_SHIFTS(method,tangle,x_init,z_init)

  !! X1 index
  if (method .eq. 1) then
    oo1(1) = (e1int1−1.)*dr1+or1 + real(iT−1.)*ds1
    oo1(2) = (e1int2−1.)*dr1+or1 + real(iT−1.)*ds1
  else
    oo1(1) = or1                          + real(iT−1.)*ds1
    oo1(2) = or1 + (dble(e1int2−1.)*dr1) + real(iT−1.)*ds1
  end if

  oo1(1)  = max(oo1(1),ox1)
  oo1(2)  = min(oo1(2),(nx1−1.)*dx1+ox1)
  iix1(1) = ( (oo1(1)−vox1(1))/dx1) + 1.5
  iix1(2) = ( (oo1(2)−vox1(1))/dx1) + 1.5
  iix1(1) = max(1,min(iix1(1),nx1))
  iix1(2) = max(1,min(iix1(2),nx1))

  !! X2 index
  if (method .eq.1 ) then
    if (ne2 .eq. 1) then
      oo2(1) = (is2−1.)*ds2 + os2 + or2
      oo2(2) = (is2−1.)*ds2 + os2 + or2
    else
      oo2(1) = (is2−1.)*ds2 + os2 + or2
      oo2(2) = (is2−1.)*ds2 + os2 + or2 + (nr2−1.)*dr2
    end if
  else
    if (ne2 .eq. 1) then
      oo2(1) = os2+(is2−1.)*ds2 + or2
      oo2(2) = os2+(is2−1.)*ds2 + or2
    else
      oo2(1)=−abs(r0)*cosh(tline(ne3))*cos(rline(1  ))+os2+(is2−1.)*ds2
      oo2(2)=−abs(r0)*cosh(tline(ne3))*cos(rline(ne2))+os2+(is2−1.)*ds2
    end if
  end if

  !!. . Don't fall off grid
    oo2(1)  = max(oo2(1),ox2)
    oo2(2)  = min(oo2(2),(nx2-1.)*dx2+ox2)
    iix2(1) = floor  ( (oo2(1)-vox2(1))/dx2) + 1
    iix2(2) = ceiling( (oo2(2)-vox2(1))/dx2)
    iix2(1) = max(1,min(iix2(1),nx2))
    iix2(2) = max(1,min(iix2(2),nx2))

 end subroutine Get_Img_Indicies
 !-----------------------------------------------------------------
 !! . . Taper Initialization Program
 subroutine rwetaper_init()
   integer :: i1,i2,j1,j2,ie1,ie2,i,jtap
   do ie2=1,ne2
      do ie1=1,ne1
         tap(ie1,ie2)=1.
      end do
   end do
   if (nxtap .gt. 1 .and. nytap .gt. 1) then

      do i2=1,nytap
         do i1=1,nxtap
            j1 = abs(nxtap-i1)
            j2 = abs(nytap-i2)
            tap(   i1 , i2 ) = cos(pi/2.* j1/(nxtap-1))*cos(pi/2.* j2/(nytap-1))
         end do
      end do

      do i2=1,nytap
         do i1=1,nxtap
            j1 = abs(nxtap-i1)
            j2 = abs(nytap-i2)
            tap(ne1-i1+1,i2) = cos(pi/2.* j1/(nxtap-1))*cos(pi/2.* j2/(nytap-1))
         end do
      end do

      do i2=1,nytap
         do i1=1,nxtap
            j1 = abs(nxtap-i1)
            j2 = abs(nytap-i2)
            tap(   i1  , ne2-i2 +1) = cos(pi/2.* j1/(nxtap-1))*cos(pi/2.* j2/(nytap-1))
         end do
      end do

      do i2=1,nytap
         do i1=1,nxtap
            j1 = abs(nxtap-i1)
            j2 = abs(nytap-i2)
            tap(ne1-i1+1,ne2-i2+1) = cos(pi/2.* j1/(nxtap-1))*cos(pi/2.* j2/(nytap-1))
         end do
      end do

      do i2=1,nytap
         do i1=nxtap+1,ne1-nytap
            j2 = abs(nytap-i2)
            tap(i1,i2)          = cos(pi/2.* j2/(nytap-1))
         end do
      end do

      do i2=1,nytap
         do i1=nxtap+1,ne1-nytap
            j2 = abs(nytap-i2)
            tap(i1,ne2-i2+1) = cos(pi/2.* j2/(nytap-1))
         end do
      end do

      do i2=nytap+1,ne2-nytap
         do i1=1,nxtap
            j1 = abs(nytap-i1)
            tap(i1,i2)          = cos(pi/2.* j1/(nxtap-1))
         end do
      end do

      do i2=nytap+1,ne2-nytap
         do i1=1,nxtap
            j1 = abs(nytap-i1)
            tap(ne1-i1+1 ,i2) = cos(pi/2.* j1/(nxtap-1))
         end do
      end do

   else if (nxtap .gt. 1 .and. nytap .le. 1) then

      do i1=1,nxtap
         j1 = abs(nxtap-i1)
         tap(   i1 , : ) = cos(pi/2.* j1/(nxtap-1))
      end do

      do i1=1,nxtap
         j1 = abs(nxtap-i1)
         tap(ne1-i1+1,:) = cos(pi/2.* j1/(nxtap-1))
      end do

   else if (nytap .gt. 1 .and. nxtap .le. 1) then

      do i2=1,nytap
         j2 = abs(nytap-i2)
         tap(:,   i2    ) = cos(pi/2.* j2/(nytap-1))
      end do

      do i2=1,nytap
         j2 = abs(nytap-i2)
         tap(:,ne2-i2+1) = cos(pi/2.* j2/(nytap-1))
      end do

   end if


end subroutine rwetaper_init
!-----------------------------------------------------------------
!! . . Write out information for visual verification
subroutine Geometry_QC(aE1,aE2,aE3,aS1,aS2,aW,aR1,aR2,aT,it,is2)
  type(myaxis)                 :: aE1,aE2,aE3,aS1,aS2,aW,aR1,aR2,aT
  integer :: it,is2
  real :: x_init,z_init,tangle
  character(len=128) :: name

  write(0,*) '    * GEOMETRY CHECK'

  !! Coordinate tilt angle
  tangle = acos(-1.)*((iT-1.)*dT+oT)/180.

  call COORD_SHIFTS(method,tangle,x_init,z_init)

  if (method .eq. 1) then
     write(0,'(a,2f11.2,2f11.2)') '    * X-dir',or1 + real(is2-1.)*ds2,(nr1-1.)*dr1+or1 + real(it-1.)*ds1
     write(0,'(a,2f11.2,2f11.2)') '    * Y-dir',or2+(is2-1.)*ds2+os2,(nr2-1.)*dr2+or2+(is2-1.)*ds2+os2
     write(0,'(a,2f11.2,2f11.2)') '    * Z-dir',oe3,(ne3-1.)*de3+oe3
  else
```

```fortran
      write(0,'(a,2f11.2,2f11.2)') '     * X-dir',    &
      or1*cos(tangle) - &
      abs(r0)*sinh(tline(ne3))*sin(rline(ne2))*sin(tangle) - x_init + real(it-1.)*ds1,&
      (dble(ne1-1.)*dr1+or1) *cos(tangle) - &
      abs(r0)*sinh(tline(ne3))*sin(rline(ne2))*sin(tangle) - x_init + real(it-1.)*ds1
      if (ne2 .eq. 1) then
         write(0,'(a,2f11.2,2f11.2)') '     * Y-dir', &
         or2+os2+(is2-1.)*ds2, &
         or2+os2+(is2-1.)*ds2
      else
         write(0,'(a,2f11.2,2f11.2)') '     * Y-dir', &
         -abs(r0)*cosh(tline(ne3))*cos(rline(1   ))+(is2-1.)*ds2+os2, &
         -abs(r0)*cosh(tline(ne3))*cos(rline(ne2))+(is2-1.)*ds2+os2
      end if
      write(0,'(a,2f11.2,2f11.2)') '     * Z-dir',&
      -(dble(ne1-1.)*dr1+or1)*abs(sin(tangle))+abs(r0)*sinh(tline( 1))*cos(tangle)-z_init,&
      (dble(ne1-1.)*dr1+or1)*sin(tangle)+abs(r0)*sinh(tline(ne3))*cos(tangle)-z_init
    end if
    write(0,'(a,i4,2f11.2,2f11.2)') '      * RECEIVERS inline    n,o,d:',nr1,or1,dr1
    write(0,'(a,i4,2f11.2,2f11.2)') '      * RECEIVERS Xline    n,o,d:',nr2,or2,dr2
    write(0,'(a,i4,2f11.2,2f11.2)') '      * EXTRAPOLATION STEPS n,o,d:',ne3,oe3,de3
    write(0,'(a,i4,2f11.2,2f11.2)') '      * SHOTS inline       n,o,d:',aS1%n,aS1%o,aS1%d
    write(0,'(a,i4,2f11.2,2f11.2)') '      * SHOTS Xline        n,o,d:',aS2%n,aS2%o,aS2%d
    write(0,'(a,i4,2f11.2,2f11.2)') '      * FREQUENCIES        n,o,d:',aW%n,aW%o,aW%d
    write(0,'(a,i4,2f11.2,2f11.2)') '      * TILTING INFO       n,o,d:',aT%n,aT%o,aT%d

  end subroutine Geometry_QC
  !-------------------------------------------------------------------
  !! . . Write out velocity information for visual verification
  subroutine Velocity_Check(CVel,RVel)
    real :: RVel(:,:,:),CVel(:,:,:)
    write(0,*) '     * VELOCITY CHECK'
    write(0,'(a,i5,i5,i5,2f11.2,2f11.2)')          '      * Cart VEL n1,n2,n3,min,max:',size(CVel,2),size(CVel,3),size(CVel
,1),minval(CVel),maxval(CVel)
    write(0,'(a,i5,i5,i5,2f11.2,2f11.2)')          '      * RWE  VEL n1,n2,n3,min,max:',size(RVel,1),size(RVel,2),size(RV
el,3),minval(RVel),maxval(RVel)
  end subroutine Velocity_Check
  !------------------------------------------------------------------
  !! . . Zero a bunch of important arrays
  subroutine Zero_arrays(RVel,CVel,RImg,CImg,ncz,ncx,ncy,aE1,aE2,aE3)
    type(myaxis) :: aE1,aE2,aE3
    integer :: ix,iy,iz,ie1,ie2,ie3,ncz,ncx,ncy
    real    :: RVel(:,:,:),CVel(:,:,:),CImg(:,:,:),RImg(:,:,:)

    !$OMP PARALLEL DO PRIVATE(ie3,ie2,ie1)
    do ie3=1,aE3%n
      do ie2=1,aE2%n
        do ie1=1,aE1%n
          RVel(ie1,ie2,ie3)=0.
        end do
      end do
    end do
    !$OMP END PARALLEL DO

    !$OMP PARALLEL DO PRIVATE(ie3,ie2,ie1)
    do ie3=1,aE3%n
      do ie2=1,aE2%n
        do ie1=1,aE1%n
          RImg(ie1,ie2,ie3)=0.
        end do
      end do
    end do
    !$OMP END PARALLEL DO

    !$OMP PARALLEL DO PRIVATE(ix,iy,iz)
    do iy=1,ncy
      do ix=1,ncx
        do iz=1,ncz
          CVel(iz,ix,iy)=0.
        end do
      end do
    end do
    !$OMP END PARALLEL DO

    !$OMP PARALLEL DO PRIVATE(ix,iy,iz)
    do iy=1,ncy
      do ix=1,ncx
        do iz=1,ncz
          CImg(iz,ix,iy)=0.
        end do
      end do
    end do
    !$OMP END PARALLEL DO

  end subroutine Zero_arrays
end module ConicalRWE3D_wemig
```