```fortran
module ConicalKernel3D
  use sep

  implicit none

  !! . . Define kernel parameter type
  type ker_par_type
     integer :: nx,ny
     real    :: dx,dy,dz,fkx,dkx,fky,dky
     real    :: ax1,ay1,bx1,by1,ax2,bx2,ay2,by2,tx,ty ! for finite difference
  end type ker_par_type
  type(ker_par_type), private:: ker_par

  !! . . Define arrays
  real,    allocatable :: cax(:,:),cay(:,:),xav(:,:,:)
  complex,allocatable :: dax(:,:),rbx(:,:),rcx(:,:),ux(:,:),vx(:,:)
  complex,allocatable :: day(:,:),rby(:,:),rcy(:,:),uy(:,:),vy(:,:)
  real    ,allocatable :: KKX(:),KKY(:)

contains

  !!----------------------------------------------------------------
  !! . . Initialization routine for Kernel propagation
  subroutine shot_ker_init(nth,nx,ny,dx,dy,dz,a,b,trick)
    integer :: nx,ny,nth,i,j
    real    :: dx,dy,dz,a,b,trick,pi,a1,a2,b1,b2

    !! . . Coefficients for two-step propagation
    a1=0.040315157; b1=0.873981642
    a2=0.457289566; b2=0.222691983

    pi=acos(-1.)
    ker_par%nx=nx
    ker_par%ny=ny
    ker_par%dx=abs(dx)
    ker_par%dy=abs(dy)
    ker_par%dz=dz

    ker_par%fkx=-pi/ker_par%dx
    ker_par%fky=-pi/ker_par%dy

    if (ker_par%nx .eq. 1) ker_par%fkx=0.
    if (ker_par%ny .eq. 1) ker_par%fky=0.

    ker_par%dkx=2.0*pi/(nx*ker_par%dx)
    ker_par%dky=2.0*pi/(ny*ker_par%dy)

    ker_par%ax1=a1*dz/(2.0*ker_par%dx*ker_par%dx)
    ker_par%ay1=a1*dz/(2.0*ker_par%dy*ker_par%dy)
    ker_par%bx1=b1/(ker_par%dx*ker_par%dx)
    ker_par%by1=b1/(ker_par%dy*ker_par%dy)
    ker_par%ax2=a2*dz/(2.0*ker_par%dx*ker_par%dx)
    ker_par%ay2=a2*dz/(2.0*ker_par%dy*ker_par%dy)
    ker_par%bx2=b2/(ker_par%dx*ker_par%dx)
    ker_par%by2=b2/(ker_par%dy*ker_par%dy)

    ker_par%tx=trick
    ker_par%ty=trick

    allocate ( dax(nx,nth),rbx(nx,nth),rcx(nx,nth),ux(nx,nth),vx(nx,nth),cax(nx,nth))
    allocate ( day(ny,nth),rby(ny,nth),rcy(ny,nth),uy(ny,nth),vy(ny,nth),cay(ny,nth))
    allocate ( xav(nx,ny,nth), KKX(nx),KKY(ny) )

    dax=0. ; rbx=0.; rcx=0.; ux=0.; vx=0.; cax=0.
    day=0. ; rby=0.; rcy=0.; uy=0.; vy=0.; cay=0.
    xav=0.; KKX=0.; KKY=0.

    call klinemap(KKX,nx,dx,0)
    call klinemap(KKY,ny,dy,0)
    call kmap_report()

  end subroutine shot_ker_init

  !!----------------------------------------------------------------
  !! . . Propagate a wavefield one extrapolation step
  subroutine shot_ker_onestep(id,wld,Rvel,x,w,signn)
    integer :: ix,iy,id
    real    :: Rvel(:,:), x(:),w,signn,tmp
    complex :: wld(:,:,:)

    !! . . Define RWE coefficients
    do iy=1,ker_par%ny
       tmp=x(iy)*w
       do ix=1,ker_par%nx
          xav(ix,iy,id)=Rvel(ix,iy)/tmp
       end do
    end do

    !! . . Split-step propagation
    call shot_ker_ssf(id,wld,w,Rvel,x,ker_par,signn)

    !! . . First FFD step
    call shot_ker_fd (id,wld,xav,ker_par,x,-signn*ker_par%ax1,ker_par%bx1,&
    -signn*ker_par%ay1,ker_par%by1)

    !! . . Second FFD step
    call shot_ker_fd (id,wld,xav,ker_par,x,-signn*ker_par%ax2,ker_par%bx2,&
    -signn*ker_par%ay2,ker_par%by2)

  end subroutine shot_ker_onestep

  !!----------------------------------------------------------------
  !! . . The split-step extrapolation part
  subroutine shot_ker_ssf(id,wld,w,Rvel,x,ker_par,signn)
    type(ker_par_type) :: ker_par
    integer :: ix,iy,id
    real    :: w,Rvel(:,:),x(:),signn,phsft,tmp
    complex :: wld(:,:,:)

    !! . . Apply appropriate w-x phase-shift
    do iy=1,ker_par%ny
       tmp = -signn * ker_par%dz * w * x(iy)
       do ix=1,ker_par%nx
          phsft = tmp / Rvel(ix,iy)
          wld(ix,iy,id) = wld(ix,iy,id) * cmplx(cos(phsft),sin(phsft))
       end do
    end do

  end subroutine shot_ker_ssf

  !!----------------------------------------------------------------
  !! . . Define which parameters we're looking at
  subroutine shot_ker_fd(id,wld,ca,ker_par,x,ax,bx,ay,by)
    type(ker_par_type) ::ker_par
    integer :: nx,ny,ix,iy,id
    real    :: ca(:,:,:),x(:),tx,ty,ax,ay,bx,by
    complex :: wld(:,:,:)

    nx=ker_par%nx
    ny=ker_par%ny
    tx=ker_par%tx
    ty=ker_par%ty

    !! . . Inline propagation (x)
    !! . . Loop over all crossline (y) indices
    if (nx .gt. 1) then
       do iy=1,ny
          ux (:,id) = wld(:,iy,id)
          cax(:,id) =  ca(:,iy,id)
!          call fd(ux(:,id),vx(:,id),tx,ax,bx,cax(:,id),nx,&
!          dax(:,id),rbx(:,id),rcx(:,id),x(iy),id)
!          wld(:,iy,id)=vx(:,id)
          call fd(ux(:,id),vx(:,id),tx,ax,bx,cax(:,id),nx,&
          dax(:,id),rbx(:,id),rcx(:,id),x(iy),id)
          wld(:,iy,id)=vx(:,id)


       end do
    end if

    !! . . Crossline propagation (y)
    !! . . Loop over all inline (x) indices
    if ( ny .gt. 1) then
       do ix=1,nx
          uy (:,id) = wld(ix,:,id)
          cay(:,id) =  ca(ix,:,id)
          call fd(uy(:,id),vy(:,id),ty,ay,by,cay(:,id),ny,&
          day(:,id),rby(:,id),rcy(:,id),1.,id)
          wld(ix,:,id)=vy(:,id)
!          call fd(uy(:,id),vy(:,id),ty,ay,by,cay(:,id),ny,&
!          day(:,id),rby(:,id),rcy(:,id),1.,id)
!          wld(ix,:,id)=vy(:,id)
       end do
    end if

  end subroutine shot_ker_fd

  !!----------------------------------------------------------------
  !! . . Setup the standard finite difference caller
  subroutine fd(u,v,tr,a,b,ca,nr,ra,rb,rc,filt,id)
    integer :: nr,ir,id
    real    :: ca(:),a,b,tr,signn,filt
    complex :: url,urr,ror,rol,u(:),v(:),ra(:),rb(:),rc(:)

    !! Case where ir=1
    url   = 0.
    urr   = u(2)
    ror   = cmplx( tr+b*(ca(1)*filt)**2 , a*ca(1)*filt**2 )
!    ror   = cmplx( tr+b*(ca(1)*filt)**2 , a*ca(1)*filt )
    rol   = conjg(ror)
    v (1) = u(1)+ror*(url+urr-2.*u(1))
    ra(1) = 1.-2.*rol
    rb(1) = rol
    rc(1) = 0.

    do ir=2,nr-1
```

```fortran
       url      = u(ir-1)
       urr      = u(ir+1)
       ror      = cmplx( tr+b*(ca(ir)*filt)**2 , a*ca(ir)*filt**2 )
!       ror      = cmplx( tr+b*(ca(ir)*filt)**2 , a*ca(ir)*filt )
       rol      = conjg(ror)
       v (ir  ) = u(ir)+ror*(url+urr-2.*u(ir))
       ra(ir  ) = 1.-2.*rol
       rb(ir  ) = rol
       rc(ir-1) = rol
    end do

    !! Case where ir=nr
    url      = u(nr-1)
    urr      = 0.
    ror      = cmplx( tr+b*(ca(nr)*filt)**2 , a*ca(nr)*filt**2 )
!    ror      = cmplx( tr+b*(ca(nr)*filt)**2 , a*ca(nr)*filt )
    rol      = conjg(ror)
    v (nr  ) = u(nr)+ror*(url+urr-2.*u(nr))
    ra(nr  ) = 1.-2.*rol
    rb(nr  ) = 0.
    rc(nr-1) = rol

    call thryj(ra,rb,rc,v,nr,id)

  end subroutine fd

  !!-----------------------------------------------------------------
  !! . . Tri-diagonal solver
  subroutine thryj(a,b,c,v,n,id)
    integer n,id
    complex :: a(:),b(:),c(:),v(:)
    integer :: ii

    !! . . for i=1
    b(1) = b(1)/a(1)
    v(1) = v(1)/a(1)

    !! for i=2,n-1
    do ii=2,n-1
       a(ii)= a(ii)-b(ii-1)*c(ii-1)
       v(ii)=(v(ii)-c(ii-1)*v(ii-1))/a(ii)
       b(ii)= b(ii)/a(ii)
    end do

    !! for i=n
    a(n)= a(n)-b(n-1)*c(n-1)
    v(n)=(v(n)-c(n-1)*v(n-1))/a(n)

    !! backsub
    do ii=n-1,1,-1
       v(ii)=v(ii)-b(ii)*v(ii+1)
    end do

  end subroutine thryj

  !!-----------------------------------------------------------------
  !! . . Fourier-domain phase shift
  subroutine wemig_phs_correction(id,dax,w,vmin,darg,flds)
    integer :: ikx,iky,id,nx,ny
    real    :: w,kr2,darg,vmin,kx,ky,flds(:),avX,avY
    real    :: tx,bx,ty,by,xav,yav,ytmp
    complex :: dax(:,:,:)
    real    :: w_v0,sr2,phsft,dz,a1,a2,b1,b2,wv2

    !! . . Filtering Coefficients
    a1=-0.040315157; b1=-0.873981642
    a2=-0.457289566; b2=-0.222691983
    dz=ker_par%dz
    nx=ker_par%nx
    ny=ker_par%ny

    w_v0=w/vmin
    wv2=w_v0*w_v0

    !! E1 correction factors
    xav = 1.!maxval( flds )
    avX = xav/w_v0

    !! E2 correction factors
    yav = 1.
    avY = yav/w_v0

    !! . . Loop over all wave-numbers
    do iky=1,ny
       ytmp = (KKY(iky)*yav)**2
       ty = (KKY(iky)*avY)**2
       by = (KKY(iky)*avY)**2
       do ikx=1,nx

          sr2 = ( (KKX(ikx)*xav)**2 + ytmp )/wv2

          !! . . Filter out evanescent energy
          if ( sr2 .gt. 1) then
             dax(ikx,iky,id) = 0.
          else

             !! . . Coefficients
             tx = (KKX(ikx)*avX)**2
             bx = (KKX(ikx)*avX)**2

             phsft=-dz*w_v0*( &
             sqrt(1.-sr2) - &
             (1. + a1 * tx / (1. + b1 * bx) + &
                   a2 * tx / (1. + b2 * bx) + &
                   a1 * ty / (1. + b1 * by) + &
                   a2 * ty / (1. + b2 * by) ) )

             dax(ikx,iky,id)=dax(ikx,iky,id)*cmplx(cos(phsft),darg*sin(phsft))

          end if
       end do
    end do

  end subroutine wemig_phs_correction

  !!-----------------------------------------------------------------
  subroutine klinemap(k,n,d,shift)
    !! . . Set the correct wavenumbers
    real, dimension(:) :: k
    integer, intent(in):: n,shift
    real,    intent(in):: d
    integer            :: i,m

    k=0.
    if(n/=1) then
       m = n/2
       do i=1,m+1
          k(i+shift)=i-1
       end do
       do i=m+2,n
          k(i+shift)=i-n-1
       end do
    else
       k=0. !! make sure k=0 if n=1
    end if

    k = k / dble(n*d)       !! 1/length
    k = 2. * acos(-1.) * k  !! radians

  end subroutine klinemap

  !-----------------------------------------------------------------
  subroutine kmap_report()
    write(0,*)"    * Wavenumber report"
    write(0,*)'    *         [n]    [o]     [d]'
    if (size(KKX) .gt. 1) then
       write(0,'(a,i4,2f11.6)')"    *     kx:",size(KKX),KKX(1),KKX(2)-KKX(1)
    end if
    if (size(KKY) .gt. 1) then
       write(0,'(a,i4,2f11.6)')"    *     ky:",size(KKY),KKY(1),KKY(2)-KKY(1)
    end if
  end subroutine kmap_report

  !!-----------------------------------------------------------------
  !! . . Release kernel memory
  subroutine shot_ker_release
    deallocate(dax,rbx,rcx,cax,ux,vx,day,rby,rcy,cay,uy,vy,xav,KKX,KKY)
  end subroutine shot_ker_release

end module ConicalKernel3D
```