FREQUENCY DOMAIN IMPLEMENTATION OF SLANT-MIDPOINT IMAGING

Richard Ottolini

Slant-midpoint stacks are as easily migrated in the frequency domain as CDP stacks. Slant-midpoint imaging enhances selected dip ranges (other papers in this report) and assists lateral velocity estimation [0]. This paper gives the frequency domain migration and diffraction transfer functions for CDP stacked, slant stacked, unstacked data images in common midpoint coordinates. The peculiar shapes of the computed slant-midpoint point scatterer response are theoretically verified by Clayton [1]. A fast computer program for modeling or migrating slant-midpoint images is included.

A wave equation transfer function may be translated entirely into the frequency domain by Stolt's method [2,3]. Consider the one way wave equation solution in *common-midpoint-stacked* coordinates

$$P_{mig}(k_x, z, \omega) = P_{dif}(k_x, z_o, \omega) \exp iz \frac{\omega}{v} \sqrt{1 - \frac{v^2 k_x^2}{\omega^2}}$$

where k_x is midpoint wave number

$$k_z = \frac{\omega}{v} \sqrt{1 - \frac{v^2 k_x^2}{\omega^2}} \qquad \text{is depth wave number}$$

The inverse fourier transform is

$$p_{mig}(x, z, t) = \iint_{-\infty}^{\infty} d\omega dk_x e^{ik_x x - i\omega t} P_{dif}(k_x, z_o, \omega) e^{i \frac{\omega}{v} \sqrt{1 - \frac{v^2 k_x^2}{\omega^2}} z}$$

The right hand side becomes a simple inverse fourier transform by

(1) Defining $t_o=0$ and $z_o=0$ as the time and depth images

(2) Using the definition of k_z to remove the ω dependence of the integral, leaving

$$P_{mig}(x,z=vt) = \iint_{-\infty}^{\infty} \left(\frac{vk_z}{k_x^2+k_z^2}\, dk_z \right) dk_x e^{ik_x} P_{dif}\left(k_x, \omega = v\sqrt{k_x^2+k_z^2}\right) e^{ik_z z}$$

Taking the fourier transform of both sides, the frequency domain transfer function equivalent is

$$P_{mig}(k_x,k_z) = \left[\frac{vk_z}{\sqrt{k_x^2+k_z^2}} \right] P_{dif}(k_x, \omega = v\ \sqrt{k_x^2+k_z^2})$$

$P_{mig}(k_x,k_z)$	$\left[\frac{vk_z}{\sqrt{k_x^2+k_z^2}}\right]$	$P_{dif}(k_x,\omega$	$= v\sqrt{k_x^2+k_z^2})$
↑ Double fourier transform migration image.	↑ Obliquity function or scaling factor.	↑ Double fourier transform migration imagion.	↑ Frequency domain mapping function.

Likewise the frequency domain transfer function for common midpoint stacked diffraction is

$$P_{dif}(k_x,\omega) = \left[\frac{\omega}{v\sqrt{\omega^2-v^2k_x^2}} \right] P_{mig}(k_x, k_z = \sqrt{\frac{\omega^2}{v^2} - k_x^2})$$

In an analogous way we derive the transfer functions for *midpoint-slant-stacked* imaging. From equation 10 of Claerbout [4], the slant-midpoint wave equation solution is

$$P_{mig}(k_x,z,\omega) = P_{dif}(k_x,z_o,\omega) \exp\ iz\ \frac{\omega}{v} \left[\sqrt{1 - \frac{(k_x - \frac{2\omega}{v}H)^2}{4\omega^2} v^2} + \sqrt{1 - \frac{(k_x + \frac{2\omega}{v}H)^2}{4\omega^2} v^2} \right]$$

where k_x is midpoint wave number as before

$H = \frac{\omega}{2v} k_h + pv$ is slant stack parameter (constant)

k_h is half offset wave number

$k_z = \frac{\omega}{v} \left[\sqrt{\ldots} + \sqrt{\ldots} \right]$ is depth wave number

Migration is

$$P_{mig}(k_x,k_z) = \frac{v}{2} \left[\frac{k_z^2(1-H^2) - 2k_x^2 k_z^2 H^2 - k_x^4 H^4}{(k_x^2+k_z^2)^{1/2} \quad (k_z^2(1-H^2) - k^2 H^2)^{3/2}} \right]$$

$$\bullet\ P_{diff}\left(k_x, \omega = \frac{vk_z}{2} \sqrt{\frac{k_x^2+k_z^2}{k_z^2(1-H^2)-k_x^2 H^2}} \right)$$

Diffraction is

$$P_{dif}(k_x,\omega) = \left[\frac{\frac{1}{v}(1-H^2)+\frac{1}{4}k_x H}{\sqrt{\omega^2(1-H^2)+\frac{\omega}{2v}k_x H-\frac{1}{4}v^2 k_x^2}} + \frac{\frac{1}{v}(1-H^2)-\frac{1}{4}k_x H}{\sqrt{\omega^2(1-H^2)-\frac{\omega}{2v}k_x H-\frac{1}{4}v^2 k_x^2}} \right]$$

$$\bullet\ P_{mig}\left(k_x, k_z = \left[\sqrt{\frac{\omega^2}{v^2}(1-H^2)+\frac{\omega}{2v}k_x H-\frac{1}{4}k_x^2} + \sqrt{\frac{\omega^2}{v^2}(1-H^2)-\frac{\omega}{2v}k_x H-\frac{1}{4}k_x^2} \right] \right)$$

Unstacked-midpoint transfer functions come from the wave equation solution given as Stolt's [3] equation 57 or Claerbout's [4] equation 7. Note they contain a third wave number dimension.

$$P_{mig}(k_x,k_h,z,\omega)=P_{dif}(k_x,k_h,z_o,\omega)\exp iz\,\frac{\omega}{v}\left(\sqrt{1-\frac{(k_x-k_h)^2v^2}{4\omega^2}}+\sqrt{1-\frac{(k\ +k_h)^2v^2}{4\omega^2}}\right)$$

Migration is

$$P_{mig}(k_x,k_h,k_z)=\frac{v}{z}\,\frac{k_z-\dfrac{k_x^2k_h^2}{k_z^2}}{\sqrt{k_x^2+k_h^2+k_z^2+\dfrac{k_x^2k_h^2}{k_z^2}}}$$

- $$P_{dif}\left(k_x,k_h,\omega=\frac{v}{2}\sqrt{k_x^2+k_h^2+\frac{k_x^2k_h^2}{k_z^2}}\right)$$

Diffraction is

$$P_{dif}(k_x,k_h,\omega)=\frac{\omega}{v}\left[\frac{1}{\sqrt{\omega^2-\frac{v}{4}(k_x^2+k_h^2-2k_xk_h}}-\frac{1}{\sqrt{\omega^2-\frac{v}{4}(k_x^2+k_h^2+2k_xk_h)}}\right]$$

- $$P_{mig}\left[k_x,k_h,k_z=\left(\sqrt{\frac{\omega^2}{v^2}-\frac{1}{4}(k_x^2+k_h^2-2k_xkh)}+\sqrt{\frac{\omega^2}{v^2}-\frac{1}{4}(k_x^2+k_h^2-2k_xk_h)}\right)\right]$$

Note the slant-midpoint-stacked and unstacked-midpoint transfer functions have the mathematical features

(1) When H or k_h are zero they reduce to the common-midpoint-stacked case.

(2) The mapping function portions (and obliquity functions) contain singularities. Nearby the singularities, the mapping functions try to select data from the nonexistant higher-than-nyquist-frequencies. Such locations may be safely zeroed out.

(3) At low frequencies the mapping function may become complex valued before it becomes imaginary. Complex and imaginary

values are attenuated and evanescent energy respectively. Though there is no computational objection to including attenuated energy, empirical results show severe distortion of the point scattering response. Attenuated energy should be zeroed out.

Documented Fortran programs for slant-midpoint migration and diffraction are attached. Minor modifications are necessary for the common-midpoint-stacked and unstacked-midpoint versions.

Figures 1 through 7 illustrate the behavior of the mapping functions on synthetic frequency domain images (kx,w and kx,kz spaces). As you recall, the mapping function is the portion of the transfer function which moves frequency values to different positions when performing frequency domain migration and diffraction. For these synthetic examples we will ignore the effects of the obliquity function, which is just a scale factor, by setting it to unity. Because positive kx values are complex conjugates of negative kx values, only half of the fourier images are displayed in the figures. Figure 1 is the kx, w source image for migrations and consists of straight lines of constant omega. Figure 2 is the slant midpoint migrated image for H = 0. Slant-midpoint for H = 0 and unstack at kh = 0 are equivalent to CDP migration. Since the CDP migration mapping function is an equation for a circle, the lines in figure 1 are transformed into the circles of figure 2. The zero regions of figure 2 represent attempts to map from non existant high omega frequencies. Figure 2 was used as the kx, kz source image for diffractions. Figure 3 is a slant-midpoint H=0 diffraction mapping where the circles of figure 2 have been mapped back into lines. The region labeled evanescent tried to map from nonexistant, imaginary kz frequencies. Figures 4 and 5 show the slant-midpoint migration and diffraction mappings for ray parameter velocity product H = pv = .3. Non zero H values no longer map lines into circles and vice-versa. Additional regions of nonexistant and attenuated energy are introduced. Figure 6 is the same diffraction mapping for H = .3 as figure 5 adding the attenuated energy. Figure 7 is the unstacked-midpoint migration mapping for cross section kh = .3. Superficially, this mapping resembles the slant mid-point migration mapping of figure 4. However, the low kx frequency content is considerably different.

Figures 8 and 9 illustrate the slant-midpoint point scatterer response. Figure 8 illustrates a suite of migrations and diffractions for different H values. Slant parameter H=0 of course generates the familar common-midpoint migration semi-circle and diffraction hyperbola. Observe for non-zero H the migration responses no longer intersect the surfaces perpendicularly. This is due to the limited dip bandwidth of slant-midpoint stacking [1]. Figure 9 shows the disasterous effect of including attenuation energy during diffraction. Scattering responses for unstacked midpoint k_h cross sections are not very enlightening.

References

[0] Lynn, W. S., "RMS Velocity Estimation in Laterally Varying Media," SEP 14

[1] Clayton, R., "Migration in Midpoint Coordinates," SEP 14

[2] Lynn, W. S., "Implementing f-k Migration and Diffraction," SEP 11

[3] Stolt, R. H., "Migration by Fourier Transform," *Geophysics*, 43:23, Feb., 1978

[4] Claerbout, J., "Migration in Slant-Midpoint Coordinates," SEP 14

FIGURE 1: Synthetic frequency domian source image for examining migration mapping functions. It consists of lines of constant omega. Only half of the kx values exhibited because of symmetry. All images are 128 by 128 points. dx = dt = v = 1.0.

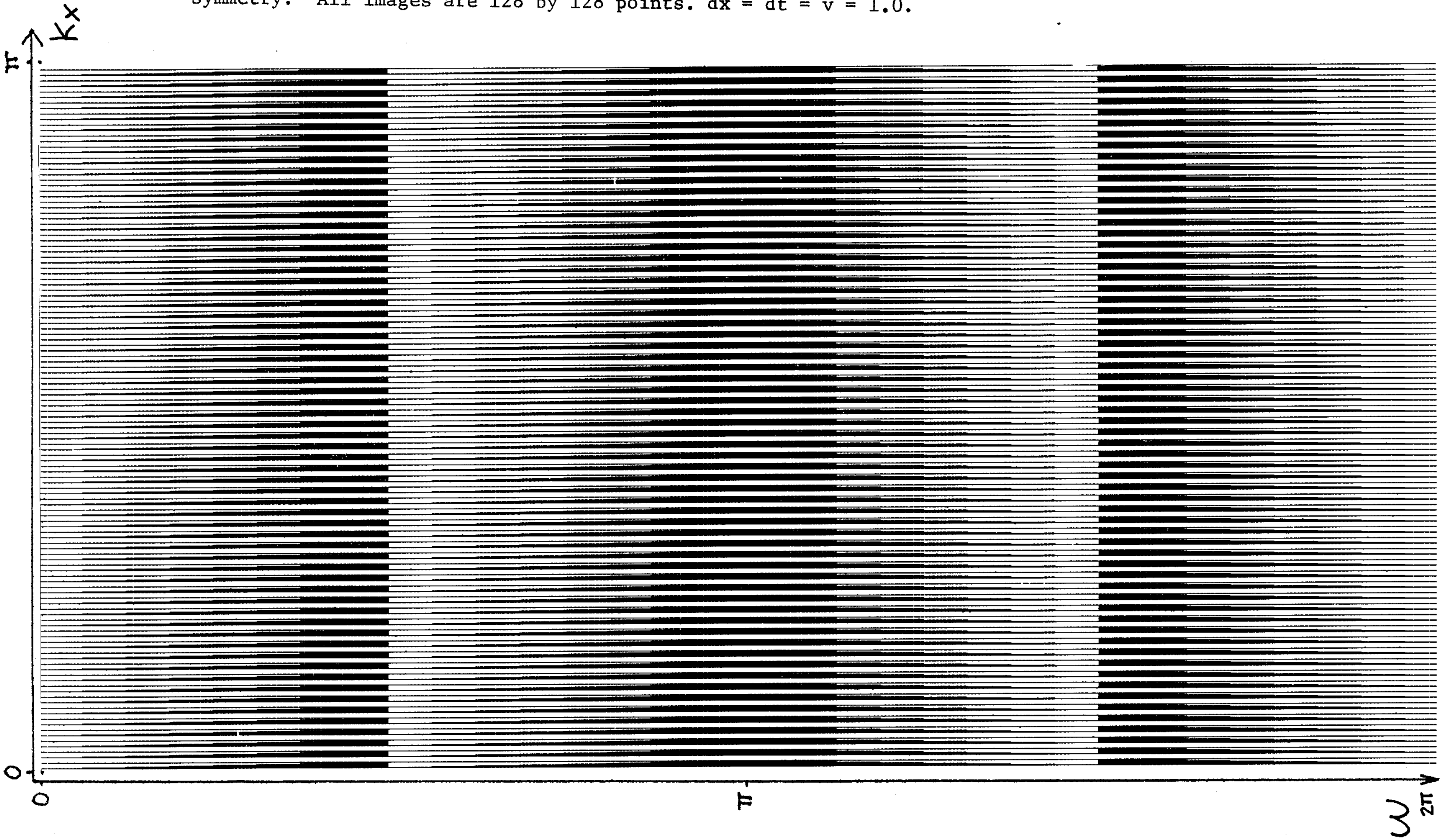


FIGURE 2: Slant-midpoint frequency domian migration mapping of figure 1 for H = 0. Lines of constant omega in figure 1 are mapped into circles. This H value is eqivalent to CDP stacked migration. Zeroed regions try to map from nonexistant, high omega values of figure 1.

This figure is also the frequency domian source image for subsequent diffractions.

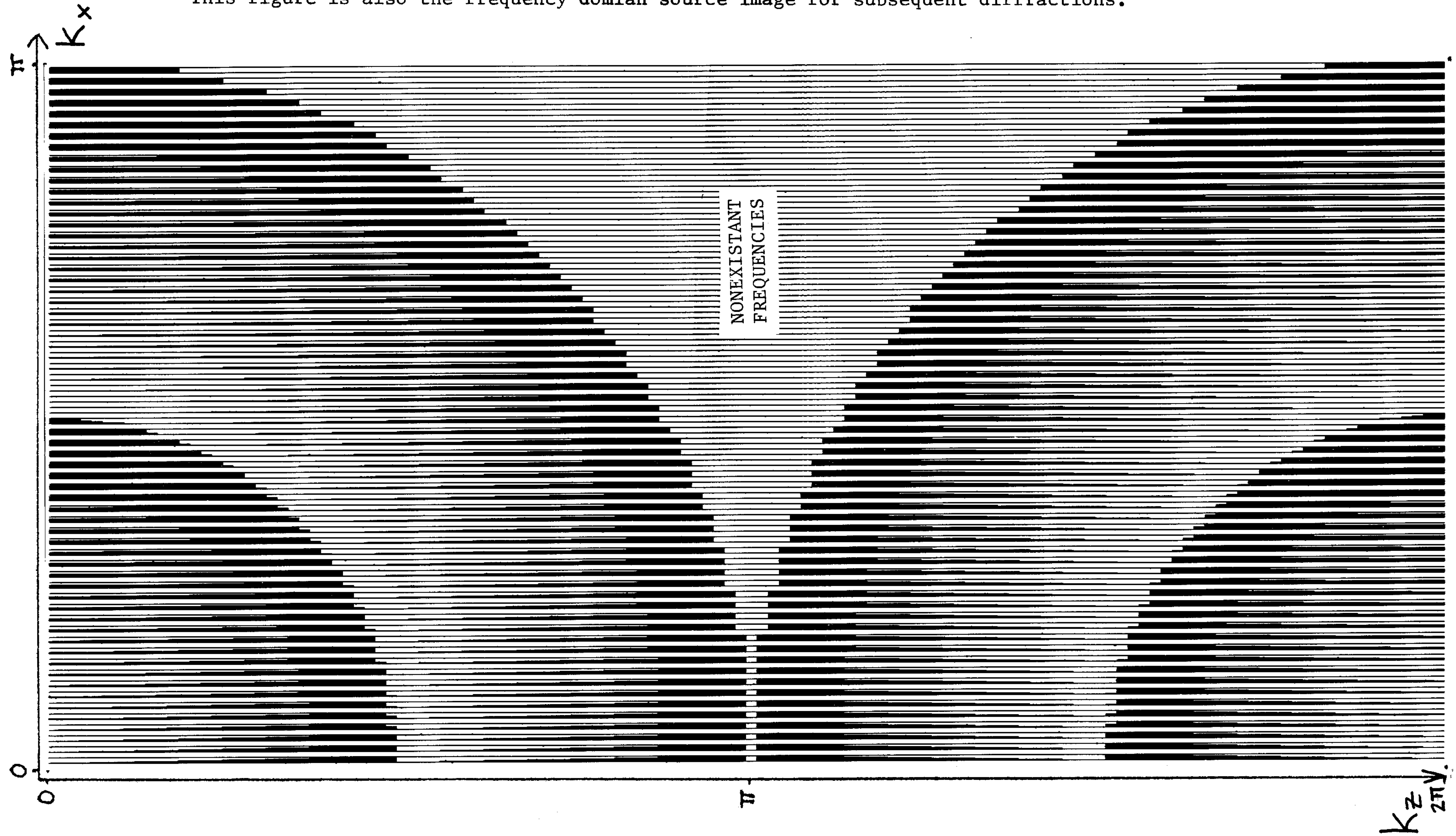
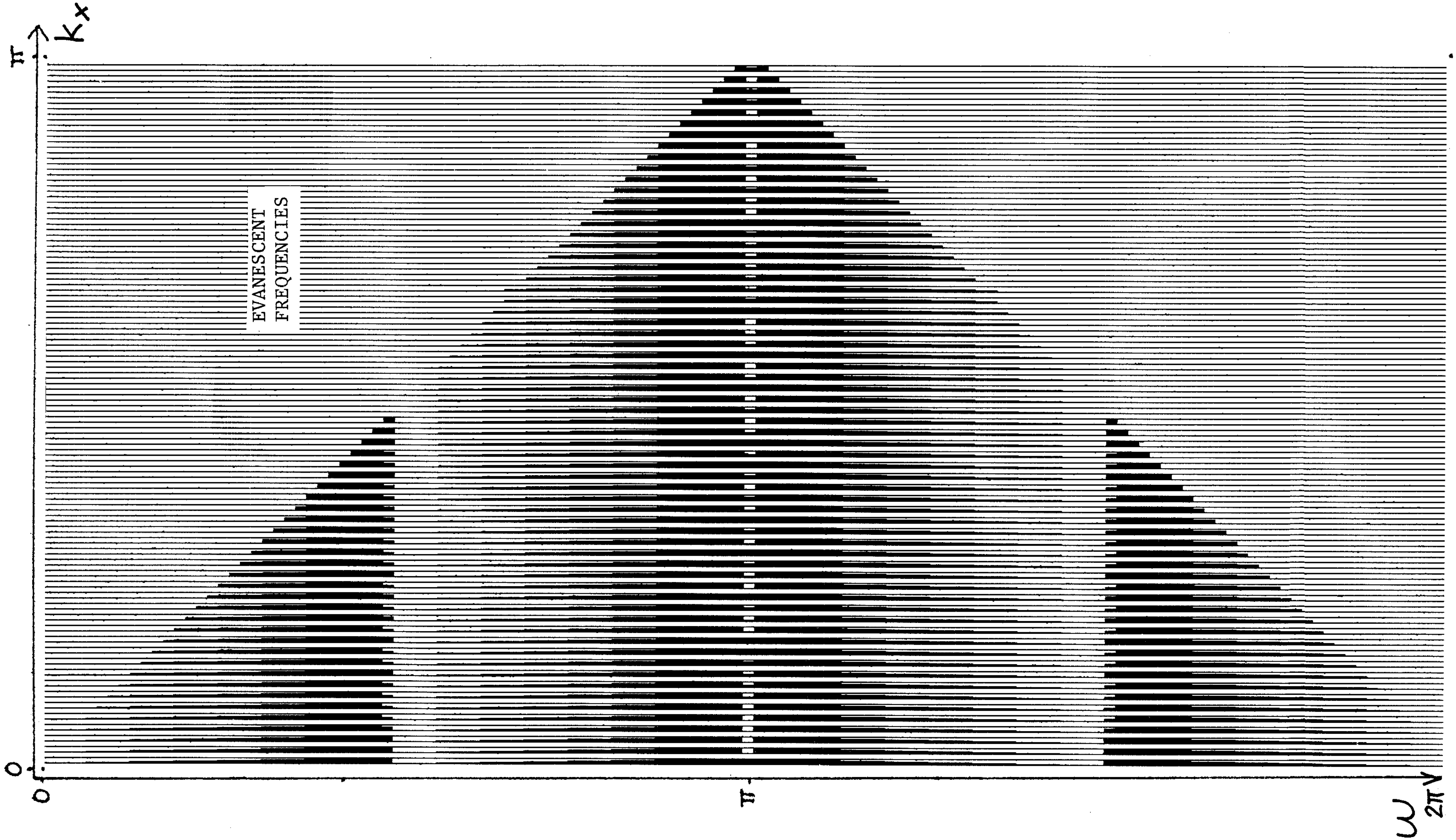


Figure 3: Slant-midpoint frequency domain diffraction mapping of figure 2. Circles have been mapped back into lines. Zeroed regions tried to map from nonexistant, imaginary kz frequencies of figure 2.



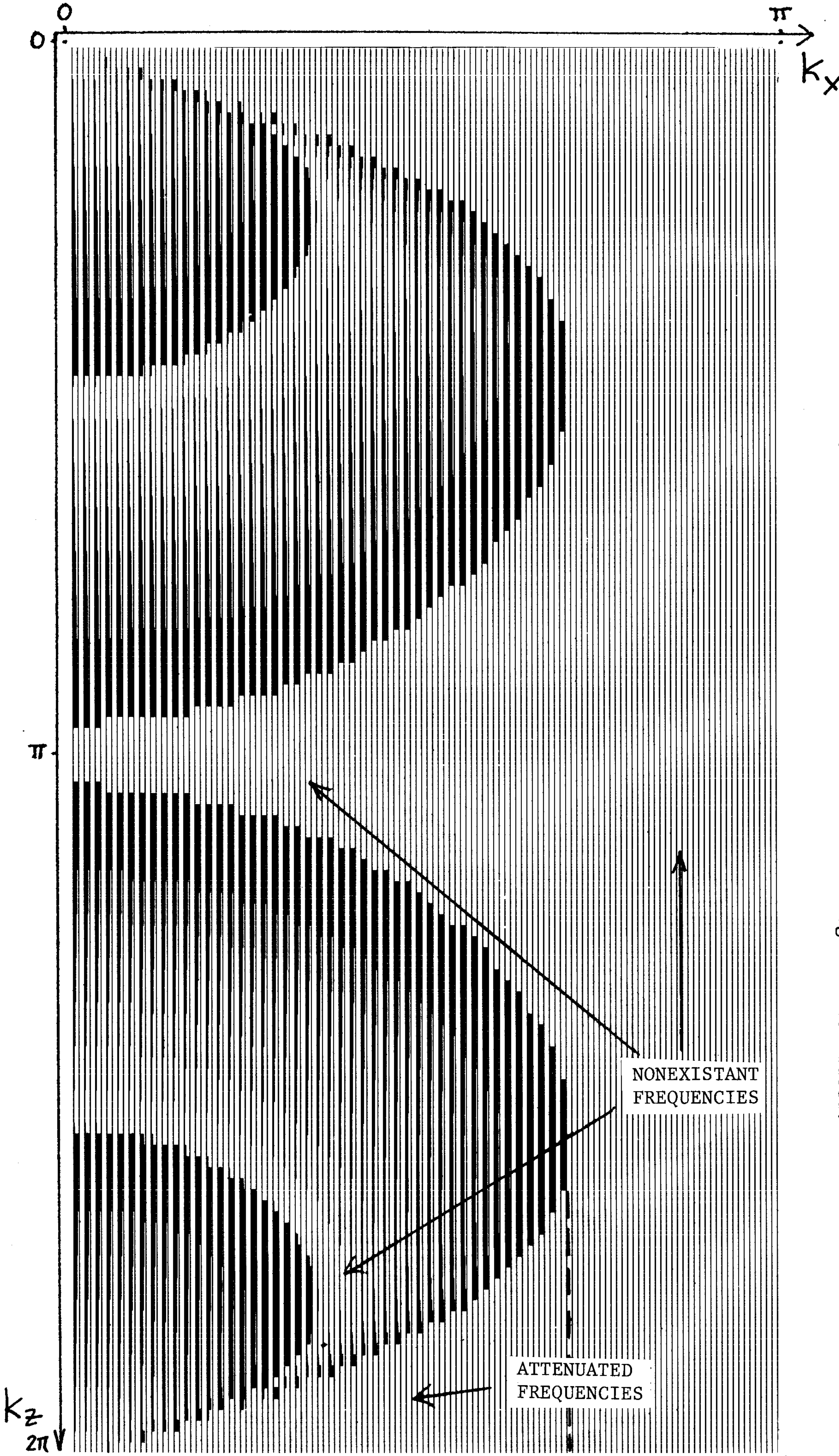


FIGURE 4: Slant-midpoint migration mapping for H = pv = .3. No longer the simple mapping of lines into circles, additional nonexistant and attenuated regions introduced.

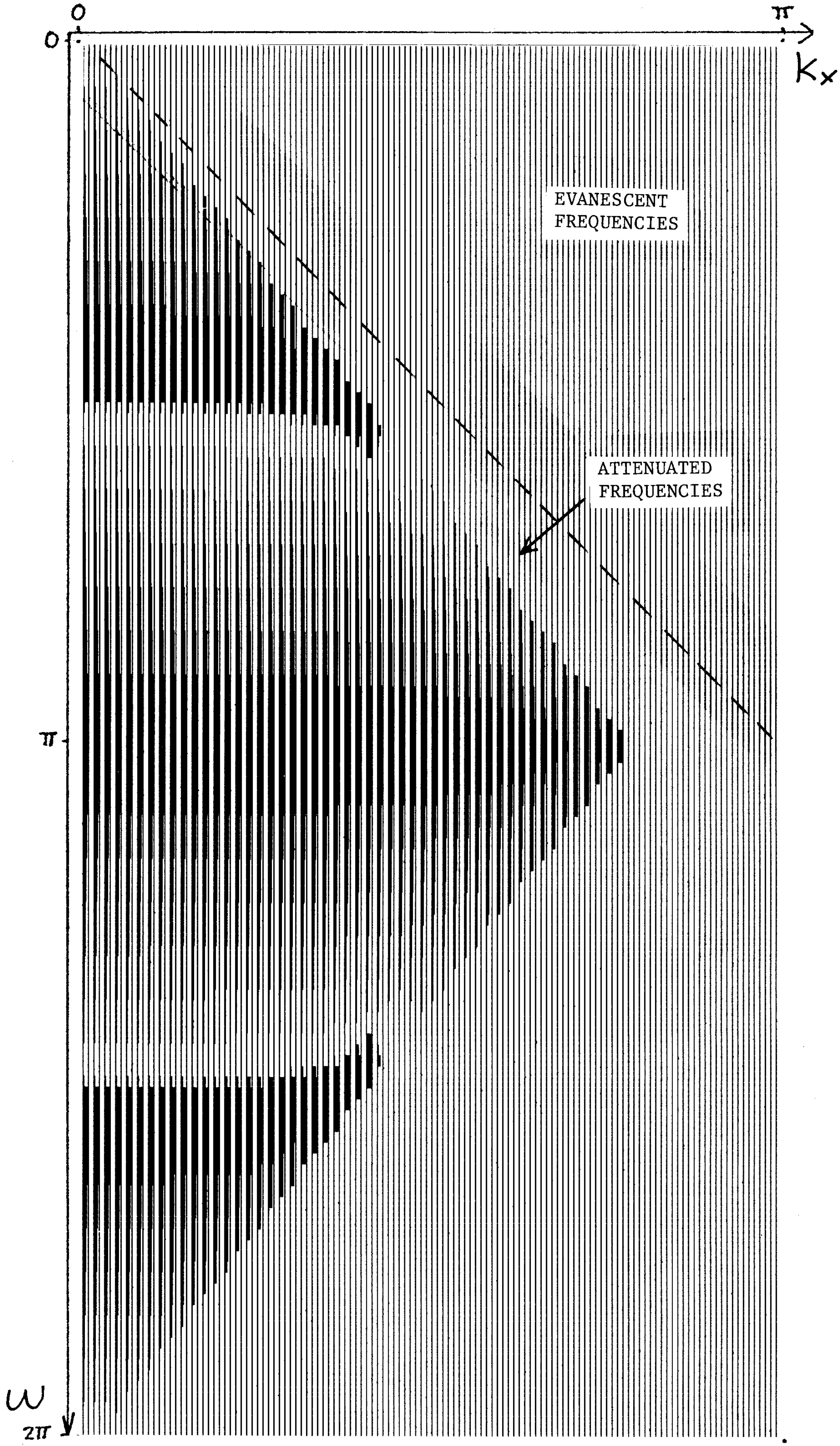


FIGURE 5; Slant-midpoint diffraction mapping for H = pv = .3. with attenuated energy zeroed.

FIGURE 6: Same as figure 5, slant-midpoint diffraction mapping for H = pv = .3., including attenuated energy.

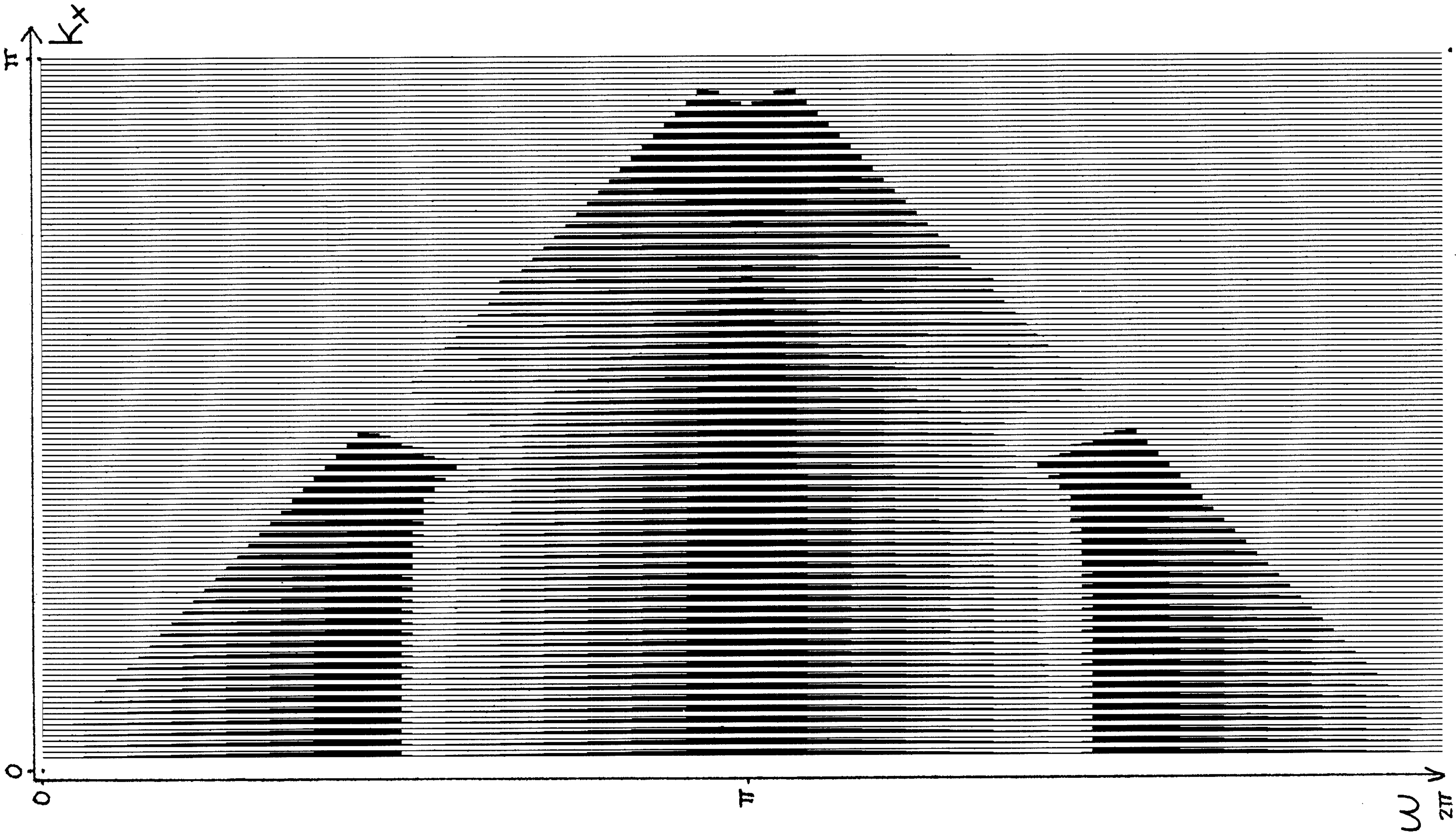
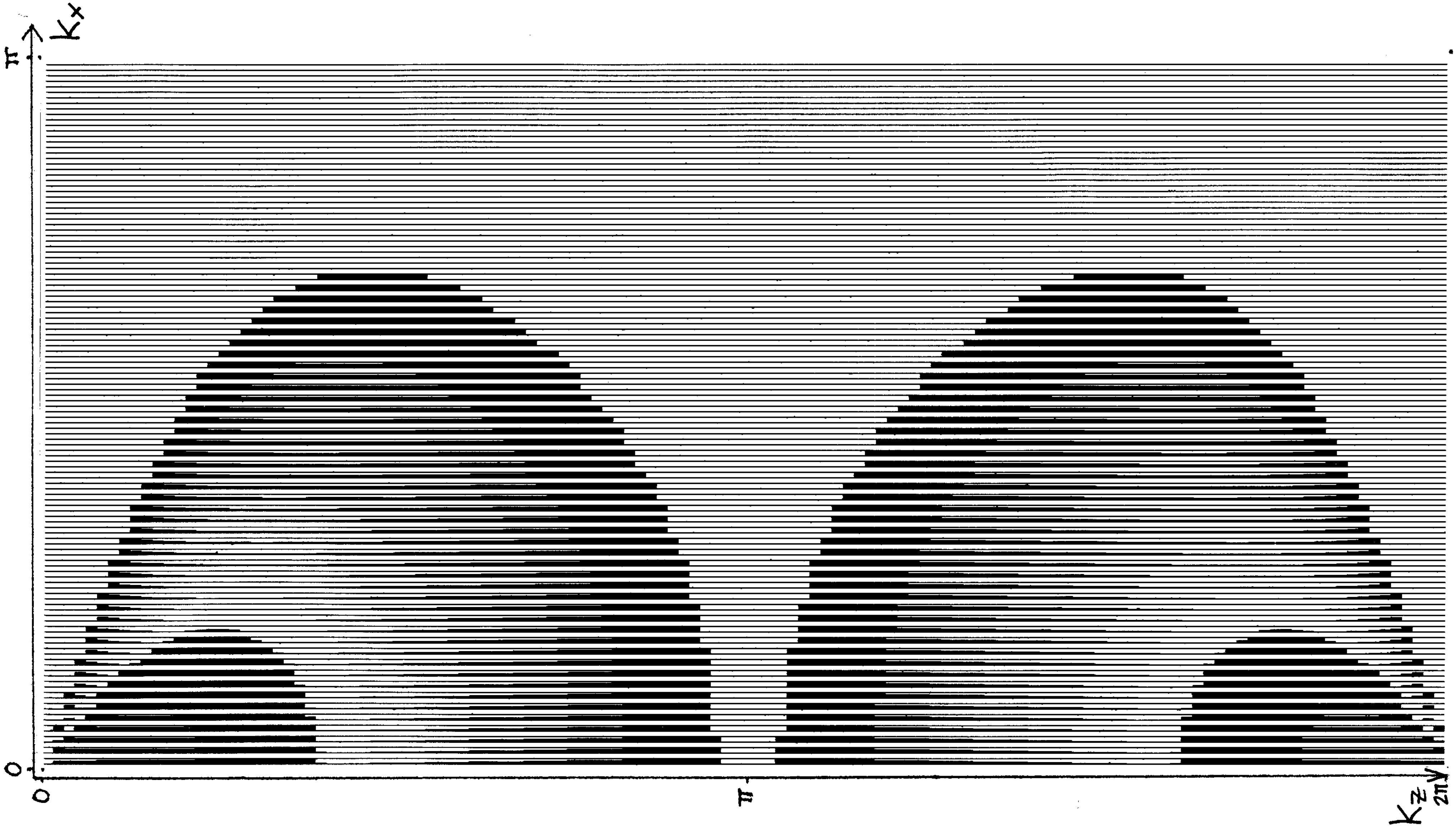


FIGURE 7: Unstacked migration mapping for cross section kh = .3. Has superficial resemblence to slant-midpoint migration mapping except considerable amplitude difference at low kx frequencies.



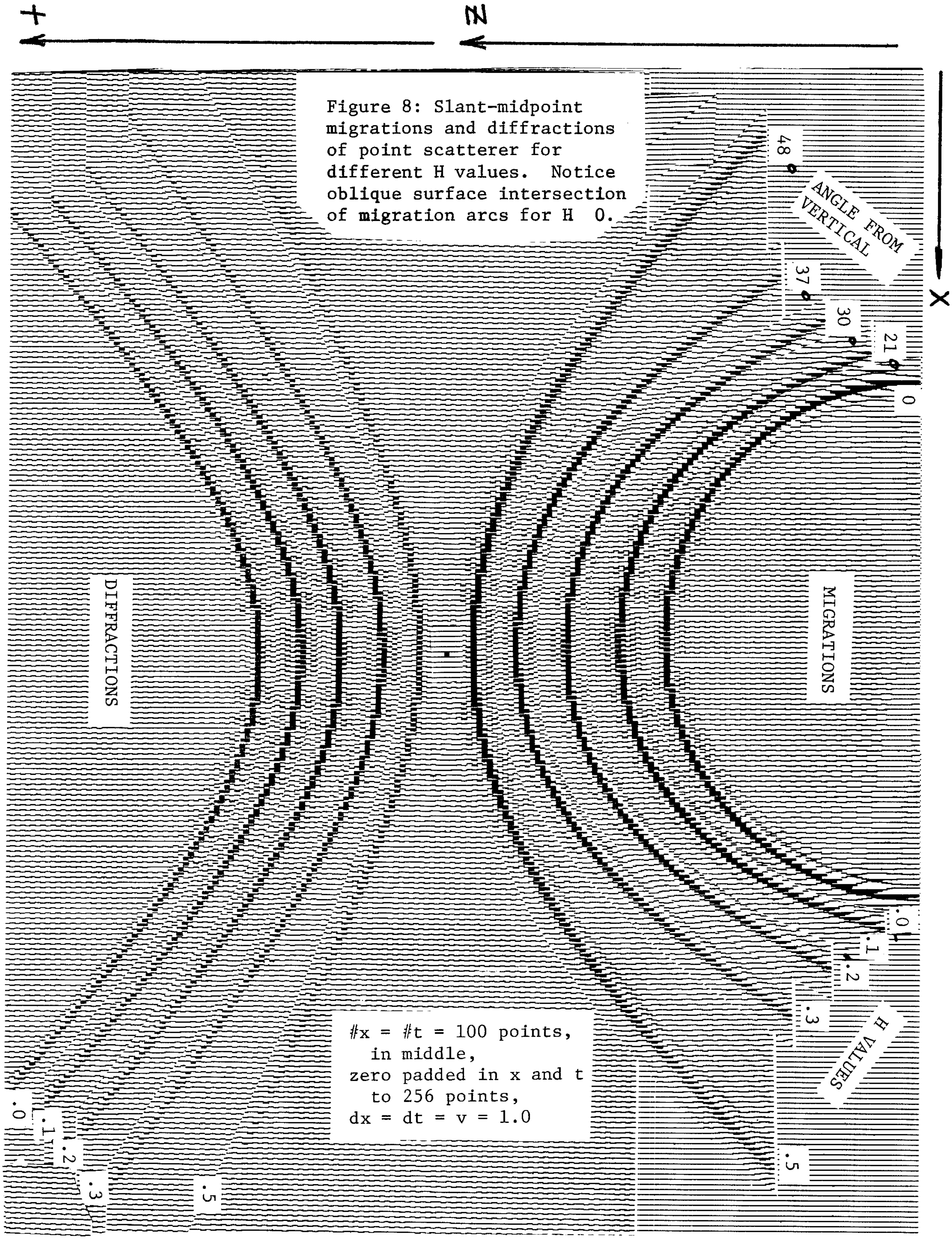
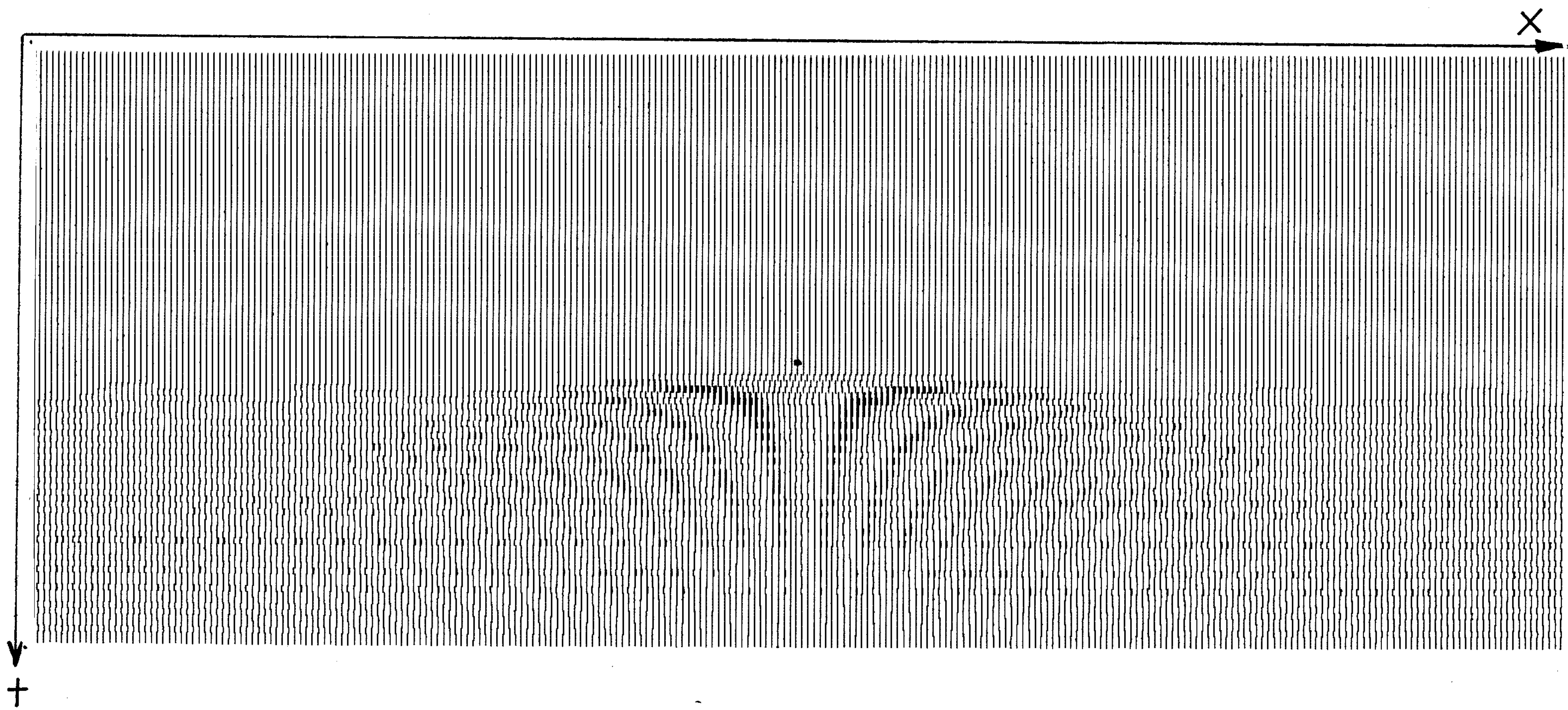


Figure 8: Slant-midpoint migrations and diffractions of point scatterer for different H values. Notice oblique surface intersection of migration arcs for H 0.

FIGURE 9: Slant-midpoint diffraction of point scatterer for H = pv = .3 including attenuated energy.



```
c       mapmigH
c
c       Author: Ottolini, Stanford Dept of Geophysics, Feb 1978
c       Machine: PDP 11/70 with FPS array processor, Princeton UNIX fortran
c
c       This program is a fast implementation of the frequency domain
c       transfer function for slant-midpoint migration:
c
c       pmig(kx,kz) = {.5*v*kz**2*(1H**2) - 2*kx**2*kz**2H**2 - kx**4*H**4}
c                 / {kx**2 + kz**2}**.5 / {kz**2*(1-H**2) - kx**2H**2}**1.5
c                 * pdiff(kx,w = .5*kz* {kx**2 + kz**2}**.5 /
c                           {kz**2*(1-H**2) - kx**2*H**2}**.5)
c
c       Program is middle of 5 step frequency domain migration program set:
c               (1) lateral fft x=>kx
c               (2) transpose to time vectors of constant kx
c               (3) inner ffts and frequency domain transfer function
c                   mapping t=>w, w=>kz, kz=>z
c               (4) transpose to kx vectors of constant z
c               (5) inverse lateral fft kx=>x
c
c       Program data:
c               infile: input filename, char*32, transposed lateral fft image,
c                       first nkx/2+1 constant kx vectors, remaining vectors
c                       redundant due to real=>complex fft symmetry
c               outfile: output file name, char*32,
                        nkx/2+1 migrated constant kx vectors
c               nt: number of time points, integer*2
c                    also number of resulting z points
c               nkx: number of constant kx vectors, power of 2, integer*2
c               nw: length of time fft, power of 2, integer*2,
c                    at least 100 greater than nt and less than 4096
c               dx: depth point seperation, real*4
c               dt: time sample rate in seconds, real*4
c               v: migration velocity in same units as dx, real*4
c               h: H value, real*4
c
c       Program variables, functions , and subroutines in order of occurrence:
c               setfil*          set device number to file name
c               nkz              number of kz values
c               nz               number od z points
c               dz               z point seperation
c               ixny             kx nyquist index
c               izny             kz nyquist index
c               nw2              w nyquist index / 2
c               pi               3.14159253
c               skx,skx2         kx scaling factors
c               skz              kz scaling factor
c               wny, wny2        w nyquist frequency
c               h1, h2           constants involving H
c               ix, xi           kx index
c               cx               kx vector
c               vclr*            clear array processor memory
c               apput*           transfer data to array processor
c               cfft*            complex to complex array processor fft
c               cfftsc*          normalize fft result in array processor
c               apget*           transfer data from array processor
c               kx, kx2          kx wavenumbers
c               izat, ziat       attenuated iz index limit
c               sqroot           modified sqrt zeros negative values
```

```
c               iznil, ixnil1   nonexistant iz index limit
c               iz, zi          kz index
c               cxbuf1,cxbuf2   buffers
c               obliq           obliquity scaling factor
c               kz, kz2         kz wavenumbers
c               num             numerator of mapping function
c               denom           denominator of mapping function
c               wi, iw          omega index
c               weight          interpolation weighting
c               cintrp          complex geometric interpolation function
c               vneg*           complex conjugate in array processor
c            * machine specific routines
c
c
c       declarations
c
        implicit complex (c)
        dimension cx(4097)
        real kx, kx2, kz2, num
c
c       program data
c
        call setfil (1,"infile",512)
        call setfil (2,"outfil",512)
        data nkx //
        data nw //
        data nkx //
        data dx //
        data dt //
        data v //
        data h //
c
c       calculate outer loop scaling and index parameters
c
c       map into same length strips
        nkz = nw
        nz = nt
        dz = v * dt
c       nyquist indices
        ixny = nkx / 2 + 1
        izny = nkz / 2 + 1
        nw2 = nw / 2
c       scale wavenumbers from seismogram parameters
        pi = 3.14159253
        skx = 2. * pi / dx / float (nkx)
        skz = 2. * / dz / float (nkz)
        wny = .5 * / v / dt
        skz2 = skz * skz
        wny2 = wny * wny
        h1 = 1. - h * h
c
c       outer loop reads and ffts (t=>w) each kx strip
c       only half of cx vectors needed because of symmetry
c
        do 80 ix = 1, ixny
                read (1) (cx(i), i=1, nt)
                call vclr (0,1,nw * 2)
                call apput (cx,0,2 * nt,2)
                call cfft (0,nw,1)
                call cfftsc (0,nw)
```

```
                call apget (cx,0,2 * nw,2)
                cx(nw + 1) = cx(1)
c
c       calculate inner loop scaling and index parameters
c       only half of mapping vector need be computed because of symmetry
c       buffering used to overwrite source vector
c
                xi = float (ix - 1)
                kx = xi * skx
                kx2 = kx * kx
                h2 = kx2 * h * h
c       attenuated energy bounds
                ziat = sqroot (h2 / h1) / skz + 1.
                izat = int (ziat)
                if (ziat.ne.float(izat)) izat = izat + 1
                if (izat.gt.izny) izat = izny
c       nonexistant energy bounds
                iznil = int (
     c          (sqroot (wny2 * h1 + wny * kx * h - .25 *kx2)
     c          + sqroot (wny2 * h1 - wny * kx * h - .25 *kx2)) / skz)+1
                iznil1 = iznil + 1
c
c       first inner loop zeros attenuated kz frequencies
c
                do 5 iz = 1, izat
                        cx(iz) = 0.
                        cx(nkz - iz + 2) = 0.
5               continue
c
c       second inner loop interpolates existant w locations
c
                if (iznil.lt.izat) goto 50
                do 50 iz = izat, iznil
                        zi = float (iz - 1)
                        kz2 = zi * zi * skz2
                        obliq = v
                        cxbuf1 = 0.
                        cxbuf2 = 0.
c       mapping index and interpolation weight
                        num = kx2 + kz2
                        denom = h1 * kz2 - h2
                        if (denom) 40,10,20
10                      if (zi.ne.0.) goto 40
                        zi = 1.
                        denom = 1.
20                      wi = zi * sqroot (num / denom) + 1.
                        iw = int (wi)
                        if (iw.gt.nw2) goto 40
                        weight = wi - float (iw)
c       obliquity factor
c                       if (num.eq.0.) goto 40
c               obliq = v * ((zi * skz) ** 4. * h1 - (2. * kz2 + h2) * h2)
c    c                  / 2. / sqroot (num * denom * denom * denom)
c       map and interpolate
30                      cxbuf1 = cintrp (weight,cx(nw-iw+2),cx(nw-iw+1)) * obli
                        cxbuf2 = cintrp (weight,cx(iw),cx(iw + 1)) * obliq
c       overwrite source vector
40                      cx(iz) = cxbuf1
                        cx(nkz - iz + 2) = cxbuf2
50              continue
```

```
c
c       third inner loop zeros non existant energy
c
                if (iznil1.gt.izny) goto 70
                do 70 iz = iznil, izny
                        cx(iz) = 0.
                        cx(nkz - iz + 2) = 0.
70              continue
c
c       outer loop inverse ffts (kz=>z) and writes kx strip
c       take vector congugates before and after fft to correct polarity
c
                call apput (cx,0,2 * nkz,2)
                call vneg (1,2,1,2,nkz)
                call cfft (0,nkz,-1)
                call vneg (1,2,1,2,nz)
                call apget (cx,0,2 * nz,2)
                write (2) (cx(i),i = 1, nz)
80      continue
        stop
        end
c
c       functions
c
c       modified square root zeros evanescent values
c
        function sqroot(arg)
        if (arg.le.0.) sqroot = 0.
        if (arg.gt.0.) sqroot = sqrt (arg)
        return
        end
c
c       complex geometric interpolation - line 2 (Lynn, SEP 11)
c
        function cintrp (weight,ca,cb)
        implicit complex (c)
        real cabs
        if (cabs(ca).ne.0.) goto 1
        cintrp = weight * cb
        return
1       if (cabs(cb).ne.0.) goto 2
        cintrp = (1. - weight) * ca
        return
2       cintrp = ca * cexp (weight * clog (cb / ca))
        return
        end
```

```
c       mapdifH
c
c       This program is a fast implementaion of the frequency domain transfer
c       function for slant-midpoint diffraction:
c
c       pdiff(kx,w) = [{(1-H**2) + .25*kx*H} / v /
c                         sqrt(w**2*(1-H**2) + w/2/v*kx*H -.25v**2*kx**2)
c                     + {(1-H**2) - .25*kx*H} / v /
c                         sqrt(w**2*(1-H**2) - w/2/v*kx*H -.25v**2*kx**2)]
c                     * pmig(kx,kz =
c                         [sqrt(w**2/v**2*(1-H**2) + w/2/v*kx*H - .25*kx**2)
c                         + sqrt(w**2/v**2*(1-H**2) - w/2/v*kx*H - .25*kx**2)]
c
c       Additional program variables from migration version:
c               iwny            omega nyquist index
c               swv             omega scaling factor / velocity
c               fac1,fac2,fac3  loop constants
c               smap            obliquity facter scaling
c               iwev            evanescent omega index limit
c
c       core of diffraction program:
c
c       calculate outer loop scaling and index parameters
c
c       map into same vector size
        nt = nz
        nw = nkz
        dt = dz / v
c       nyquist frequency indices
        ixny = nkx / 2 + 1
        iwny = nw / 2 + 1
c       compute wave number scale factors from seismogram parameters
        pi2 = 2. * 3.14159253
        skx = pi2 / dx / float (nkx)
        skz = pi2 / dz / float (nkz)
        swv = .5 * pi2 / v / dt / float (nw)
        fac1 = swv * swv * (1 - h * h)
        smap = (1 - h * h) / v
c
c       outer loop reads and ffts (z=>kz) each kx strip
c       only half of the kx strips need be mapped because of symmetry
c
        do 30 ix = 1, ixny
                read (1) (cx(i),i = 1,nz)
                call vclr (0,1,nkz * 2)
                call apput (cx,0,2 * nz,2)
                call cfft (0,nkz,1)
                call cfftsc (0,nkz)
                call apget (cx,0,2 * nkz,2)
                cx(nkz + 1) = cx(1)
c
c       calculate inner loop parameters
c
                kx = float (ix - 1) * skx
                kx2 = kx * kx
                fac2 = swv * h * kx
                fac3 = -.25 * kx2
                smap1 = smap + .25 * kx * h
                smap2 = smap - .25 * kx * h
c       evanescent index bounds
```

```
            iwev = iwny - int (kx * (1. + h) / 2. / swv)
c
c       first inner loop maps w for nonevanescent kz points
c       only half of the mapping vector need be computed because of
c       symmetry
c       the source vector may be overwritten by the mapped vector by
c       going from high to low frequencies and using double buffering
c
            if (iwev.lt.1) goto 15
            cxbuf1 = cx(iwny + 1)
            cxbuf2 = cx(nw - iwny + 1)
            do 10 iw = 1, iwev
                wi = float (iwny - iw)
c       mapping index and interpolation weight
                kz1 = sqroot (fac1 * wi * wi + fac2 * wi + fac3)
                kz2 = sqroot (fac1 * wi * wi - fac2 * wi + fac3)
                zi = (kz1 + kz2) / skz + 1.
                iz = int (zi)
                weight = zi - float (iz)
c       obliquity factor
                obliq = 1. / v
c               if (kz1.gt.0.) obliq = smap1 / kz1
c               if (kz2.gt.0.) obliq = obliq + smap2 / kz2
c       map and interpolate
            cxbuf3 = cintrp (weight,cx(nkz-iz+2),cx(nkz-iz+1))*obliq
                cxbuf4 = cintrp (weight,cx(iz),cx(iz+1))*obliq
c       overwrite source vector
                cx(iwny - iw + 2) = cxbuf1
                cx(nw - iwny + iw) = cxbuf2
                cxbuf1 = cxbuf3
                cxbuf2 = cxbuf4
10          continue
            cx(iwny - iwev +1) = cxbuf3
            cx(nw - iwny + iwev + 1) = cxbuf4
c
c       second loop zeros evanescent w points
c
15          iwev = iwny - iwev
            if (iwev.gt.iwny) iwev = iwny
            if (iwev.lt.1) goto 20
            do 20 iw = 1, iwev
                cx(iw) = 0.
                cx(nw - iw + 2) = 0.
20          continue
c
c       outer loop inverse ffts (w=>t) and writes kx strip
c       take vector conjugates before and after fft to correct polarity
c
            call apput (cx,0,2 * nw,2)
            call vneg (1,2,1,2,nw)
            call cfft (0,nw,-1)
            call vneg (1,2,1,2,nt)
            call apget (cx,0,2 * nt,2)
            write (2) (cx(i),i = 1,nt)
30      continue
```