Chapter 9

Hyperbola tricks

PIXEL-PRECISE VELOCITY SCAN

In SEP-61 page 361 I introduced a radical new method of transformation to velocity space. Here we'll evaluate it.

Traditionally velocity scan is done by the loop structure given there, namely:

```
do v
    do tau
        do x
            t = sqrt( tau**2 + (x/v)**2 )
            velo( tau, v) = velo( tau, v)  +  data( t, x)
```

These loops transform distance x to velocity v something like Fourier analysis transforms time to frequency. Here we'll investigate a new alternative that gives conceptually the same result, but one that differs in practical ways. The new alternative is to transform with the loop structure:

```
do tau
    do t = tau, tmax
        do x
            v = sqrt( x**2  / ( t**2 - tau**2 ))
            velo( tau, v) = velo( tau, v)  +  data( t, x)
```

Notice that $t = \sqrt{\tau^2 + (x/v)^2}$ in the conventional code is algebraically equivalent to $v = x/\sqrt{t^2 - \tau^2}$ in the new code. The traditional method finds one value for each point in *output* space whereas the new method uses each point of the *input* space exactly once.

The new method differs from the traditional method in cost, smoothing, accuracy, and truncation. The cost of traditional velocity scan is proportional to the product

$N_t N_x N_v$ of the lengths of the axes of time, offset, and velocity. The cost of the new transform is proportional to the product $N_t^2 N_x/2$. Normally $N_t/2 > N_v$ so the new alternative is somewhat more costly than traditional velocity scan, but not immensely so, and in return we can have all the (numerical) resolution we wish in velocity space at no extra cost. The verdict is not in yet on whether the new method is better than the old method in routine practice but the reasoning behind the new method teaches many lessons. Not examined here is the smooth envelope (chapter 10) that is generally a post-process to conventional velocity scan but may be integrated with the scan in some implementations.

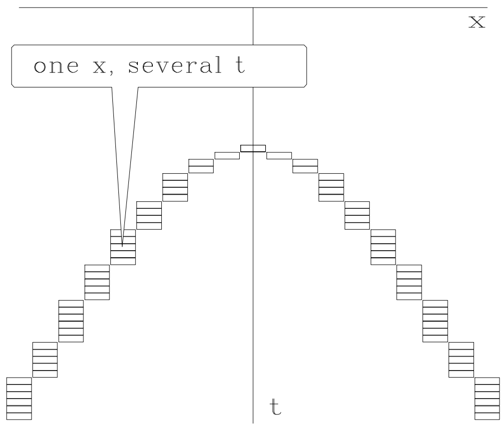

Figure 1: A typical hyperbola crossing a typical mesh. Notice that the curve is represented by multiple time points for each x. jon4-lineint [NR]

Certain facts about aliasing must be borne in mind as one defines any velocity scan. A first concern arises because typical hyperbolas crossing a typical mesh encounter multiple points on the time axis for each point on the space axis. This is shown in Figure 1. An aliasing problem will be experienced by any program that selects only one signal value for each x instead of the multiple points that are shown. The extra boxes complicate traditional velocity scan. Many programs ignore it without embarrassment only because low velocity events contain only shallow information of the earth. (A cynical view is that field operations tend to oversample in offset space because of this limitation in some velocity programs.) A significant improvement is made by summing all the points in boxes. A still more elaborate analysis (that we will not pursue here) is to lay down a hyperbola on a mesh and interpolate a line integral from the traces on either side of the line.

A second concern arises from the sampling in velocity space. Traditionally people question whether to sample velocity uniformly in velocity, slowness, or slowness squared. Difficulty arises first on the widest-offset trace. When jumping from one velocity to the next, the time on the wide-offset trace should not jump too far so as to leave a gap, as shown in Figure 2.

Under the new method there is no chance of missing a point on the wide-offset trace. For each depth τ, every point below τ in the input-data space (including the wide-offset trace) is summed exactly once into velocity space (whether that space is discretized uniformly in velocity or slowness). Also, the inner trace enters *only* once.

Under the new method many old interpolation issues are irrelevant, but new ques-

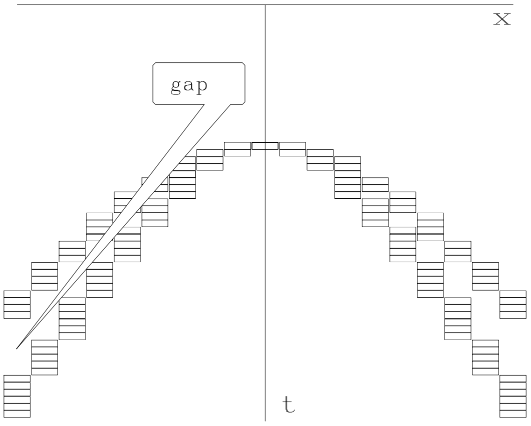

Figure 2: Too large an interval in velocity will leave a gap between the velocity scans. jon4-deltavel [NR]

tions arise. The (t, x) position of the input data is exact, as is τ. Interpolation becomes a question only on v. Since the effort is independent of the number of points in velocity, you could sample densely and use nearest-neighbor interpolation (or any other interpolation). A disadvantage is that some points in (τ, v)-space may happen to get *no* input data especially if you refine v too much.

Including some scaling that will be described later, the result of the new velocity transformation is shown in Figure 3. The code that generated Figure 3 is just like the

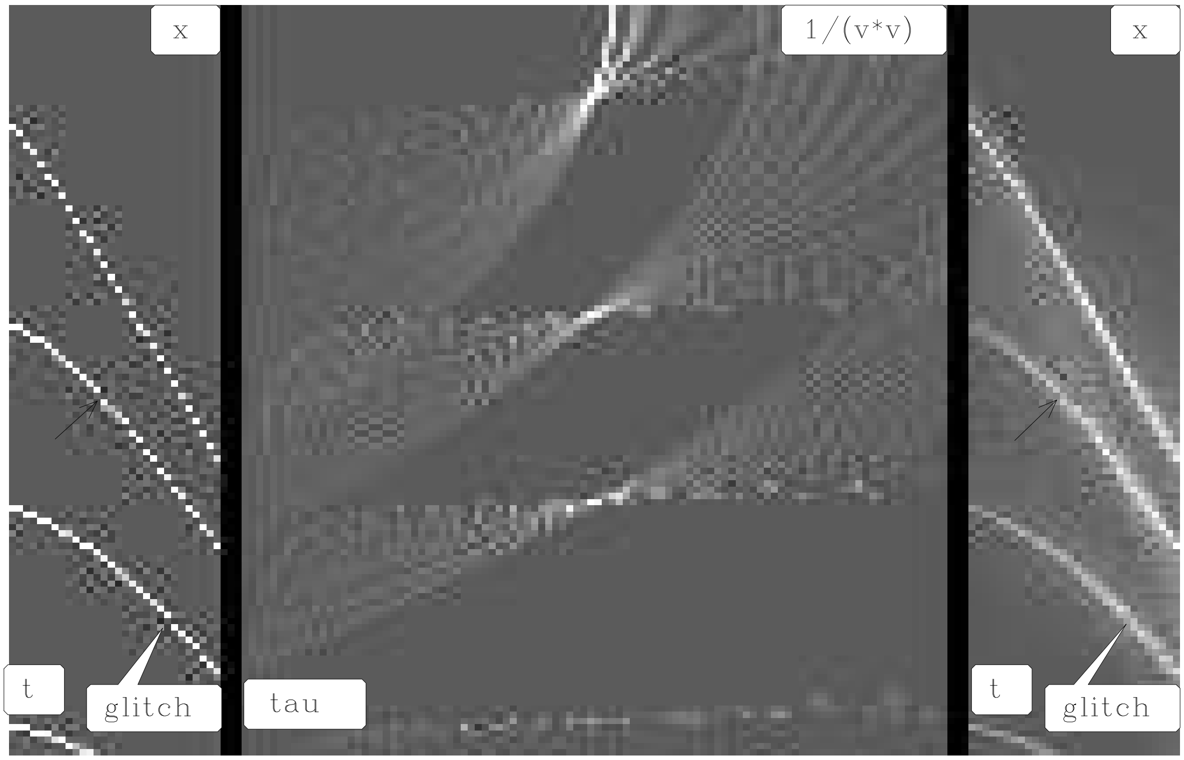

Figure 3: Offset to slowness squared and back to offset. jon4-vspray1 [NR]

pseudocode above except that it parameterizes velocity in uniform samples in inverse velocity squared, $s = v^{-2}$. A small advantage of using s-space instead of v-space is that the trajectories you see in (τ, s)-space are readily recognized as parabolas, namely $\tau^2 = t^2 + x^2 s$, where each parabola comes from a particular point in (t, x).

To exhibit all the artifacts as clearly as possible I changed all signal values to their signed square roots before plotting brightness. This has the effect of making the plots look noisier than they really are. I also chose Δt to be unrealistically granular to enable you to see each point. The synthetic input data was made with nearest neighbor NMO. Notice that resulting timing irregularities in the input are also present in the reconstruction. This shows a remarkable precision.

Balancing the pleasing result of Figure 3 is the poor result with the same program shown in Figure 4. It shows that points in velocity space map to bits of hyperbolas in offset space—not to entire hyperbolas. The figure also shows that small-offset points become dotted lines with widely separated dots in velocity space.

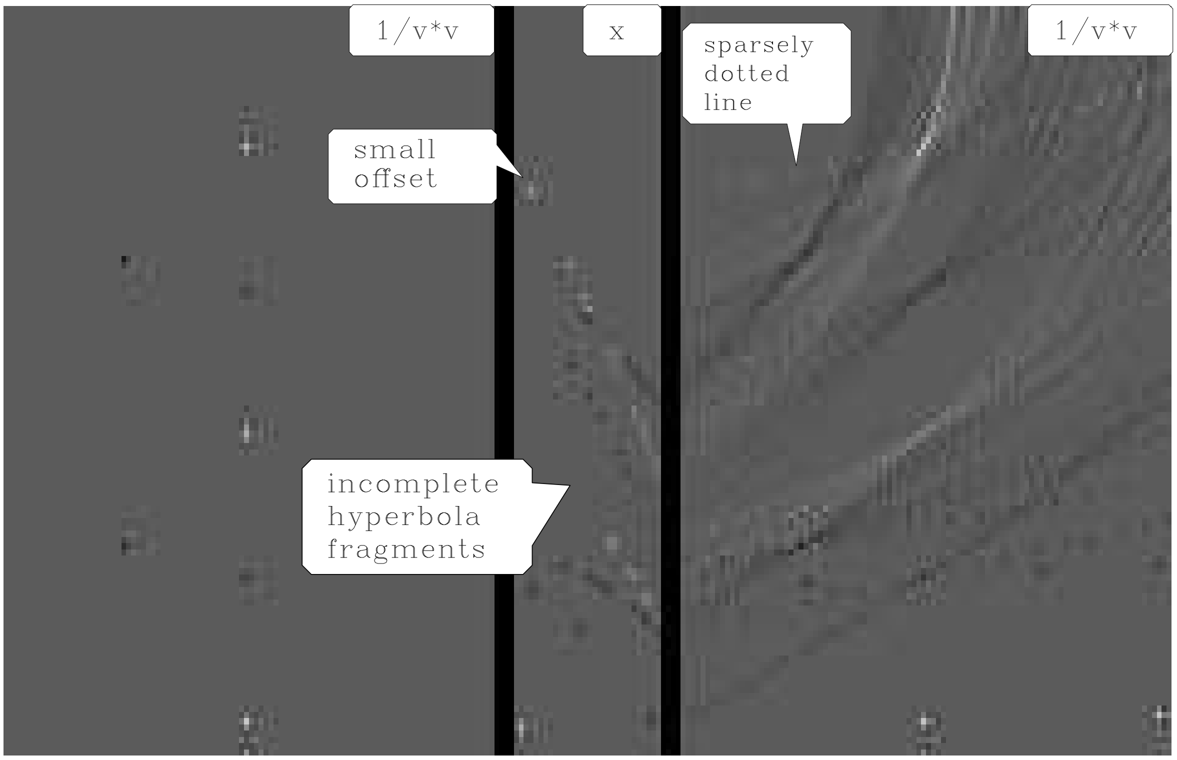

Figure 4: Sloth to offset and back to sloth. $\boxed{\text{jon4-vspray2}}$ [NR]

The problem of hyperbolas being present only discontinuously is solvable by smearing over any axis, t, x, τ, or v but we would prefer intelligent smoothing over the appropriate axis.

Smoothing in velocity

To get smoother results I took the time axis to be continuous and the signal value at (t, x) to be distributed between the two points $t_{-} = t - \Delta t/2$ and $t_{+} = t + \Delta t/2$. The two time points $t_{\pm}$ and the x-value are mapped to two slownesses $s_{\pm}$. The signal from the (t, x)-pixel is sprayed into the horizontal line $(\tau, s_{\pm})$. To enable you to reproduce the result, I include the program below:

```
subroutine vspray( conj,  nt,dt,t0,  nx,dx,x0, tx,  ns,ds,s0, zs)
integer conj, it, nt, iz, nz, ix, nx, is, ns, isp, ism
real    tx(nt,nx), zs(nt,ns), scale
real    z,dz,z0, t,dt,t0, x,dx,x0, s,ds,s0, sm,sp, xm,xp, tm,tp
nz=nt;  dz=dt;  z0=t0;
if( conj == 0 ) { do is=1,ns;  do iz=1,nz;  zs(iz,is) = 0.}
else            { do ix=1,nx;  do it=1,nt;  tx(it,ix) = 0.}
if( conj == 0)  { do ix=1,nx;  call halfdif ( 1, nt, tx(1,ix), tx(1,ix) )}
do iz=  1, nz {  z = z0 + dz*(iz-1)
do ix=  1, nx {  x = x0 + dx*(ix-1)
do it= iz, nt {  t = t0 + dt*(it-1)
        tm = t-dt/2;    xm = x
        tp = t+dt/2;    xp = x
        sm = (tm**2 -z**2)/xp**2;  ism = 1.5+(sm-s0)/ds
        sp = (tp**2 -z**2)/xm**2;  isp = 1.5+(sp-s0)/ds
        if( ism<2 ) next
        if( isp>ns) next
        scale = sqrt( t / (1.+isp-ism) ) / x
        do is= ism, isp {
                if( conj == 0)
                        zs(iz ,is) = zs(iz ,is) + tx(it ,ix) * scale
                else
                        tx(it ,ix) = tx(it ,ix) + zs(iz ,is) * scale
                }
        } } }
if( conj != 0)  { do ix=1,nx;  call halfdif ( 0, nt, tx(1,ix), tx(1,ix) )}
return; end
```

Figure 5 shows the result on the same inputs as in Figures 3 and 4.

Rho filter

Notice the dark halo around the reconstruction in Figure 3. It was suppressed in Figure 5 by the subroutine `halfdif()`. Recall that slant-stack inversion (see IEI for example) requires an $|\omega|$ filter. Without doing any formal analysis I guessed that the same filter would be helpful here because the dark halo has a strong spectral component at $\omega = 0$ which would be extinguished by an $|\omega|$ filter. Because of the close relation to wave propagation and causality, I found it appealing to factor $|\omega|$ into a causal $\sqrt{-i\omega}$ part and an anticausal $\sqrt{i\omega}$ part. I applied a causal $\sqrt{-i\omega}$

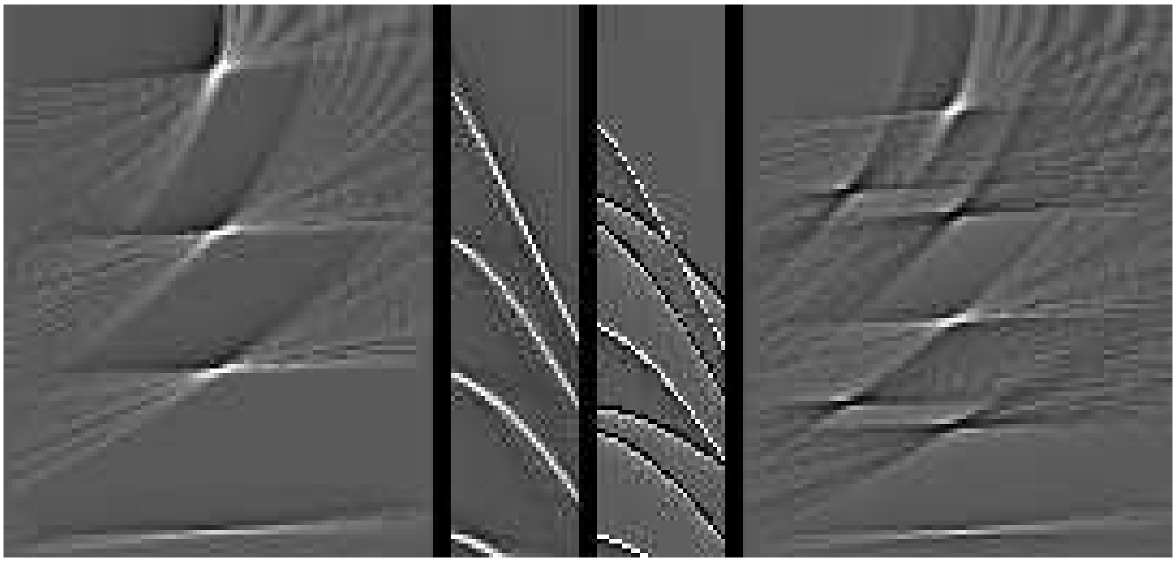

Figure 5: Horizontal line method. Compare the left to Figure 3 and the right to 4. jon4-vspray4 [NR]

after generating the (t, x)-space and an anticausal $\sqrt{i\omega}$ before making the (τ, v^{-2})-space. I implemented the causality by taking the square root of a Fourier domain representation of causal differentiation, namely $(1 - Z)^{1/2}$. The listing of this half-order differentiator is straightforward.

Notice also that `vspray()` includes a scaling variable named `scale`. I did not develop a theory for this scale factor, but if you omit it, amplitudes in the reconstructions will be far out of amplitude balance with the input.