Stanford Exploration Project, Report 75, November 17, 1997, pages 97–103

Stanford Exploration Project, Report 75, November 17, 1997, pages 97–103

Short Note

Fast anti-aliased Kirchhoff migration and modeling

Dimitri Bevc and Jon Claerbout[1]

keywords: *algorithm, modeling, synthetic data*

INTRODUCTION

We describe an optimized Kirchhoff migration and modeling routine which handles the problem of operator aliasing. Slight modifications of this algorithm could be useful for velocity analysis, pre-stack migration and DMO.

Several Kirchhoff routines are presented in SEP73. The simplest routines (Claerbout, 1992b) are tutorial and do not address the problem of operator aliasing. A more elaborate routine which overcomes the aliasing problem (Claerbout, 1992a) proves to be slow.

We incorporate the best features of the fast program `kirchfast()` (Claerbout, 1992b) and the anti-aliasing program `trimo()` (Claerbout, 1992a) to make a fast anti-aliased Kirchhoff migration and modeling routine. Timing tests on the SPARCstation IPC show the new routine to be twelve times faster than `trimo()` and only three times slower than `kirchfast()`.

IMPLEMENTATION

The fast anti-aliasing routine `kfastaax()` achieves most of its speed by re-ordering the do-loops of `aamig()` and `trimo()`. Re-ordering is done in the same way as for `kirchslow()` and `kirchfast()` so that the outer loop runs over offset. This allows overhead calculations to be reused many times. Additional speed is obtained by transposing the input and output so that the inner loop over `ix` is along stride instead of across stride. This inner loop over `ix` is performed in the linear interpolation subroutine `spot3x()`. The speed gained by transposing the input is achieved at the cost of assuming that velocity does not vary with offset.

Program listings are provided for `kfastaax()` and `spot3x()`. The subroutine `doubint()` performs double integration. For `conj=0` it integrates once causally and

[1]**email:** not available

then once anticausally.

For `conj=1` `kfastaax()` migrates the input by summing along hyperbolic trajectories with triangular smoothing in time. For `conj=0` `kfastaax()` models the data by spraying the input along hyperbolic trajectories.

For data modeling the program proceeds as follows:

- Transpose the input so that the inner loop is along stride.
- Loop over offset `ih`
- Within the `iz` loop the slope of the hyperbola is calculated and used to calculate the end points of the smoothing triangles. An amplitude factor which incorporates spherical divergence, obliquity, and triangle width is calculated.
- The triangle endpoints and amplitudes are passed to the interpolation routine `spot3x()` along with a column of the transposed input data. `spot3x()` returns six column vectors of linearly interpolated output.
- The output is transposed and integrated twice. The causal integration followed by anti-causal integration completes the triangular smoothing.

```
# Kirchhoff migration and diffraction.  (fast anti-aliased)
# combination of kirchfast() and trimo().  Written by Dimitri Bevc 082492
#
subroutine kfastaax(conj,add,anti,vrms,    t0,dt,dx,nt,nx,modl,      data)
integer             conj,add,                        nt,nx
real                              vrms(nt),t0,dt,dx,     modl(nt,nx),data(nt,nx)
real        z, t, h,       tp, tm,   amp, anti, slope
integer ix,iz,it,ih,      itp,itm,  xstart, xend
temporary real ss(nt),trdata(nx,nt),trmodl(nx,nt)

call conjnull(      conj, add,                         modl,nt*nx,  data,nt*nx)
call null(     ss,nt) ; call null(trmodl,nx*nt) ; call null(trdata,nx*nt)

if(conj!=0)
        do ix=1,nx {
                call doubint( 1,  0, nt, ss, data(1,ix))
                do it=1,nt
                        trdata(ix,it)=ss(it)
                }
if(conj==0)
        do ix=1,nx
                do it=1,nt
                        trmodl(ix,it)=modl(it,ix)
do ih= -nx, nx {        h = dx * ih                     # h = offset
        do iz= 1, nt {  z = t0 + dt * (iz-1)            # z = travel-time depth
                        t = sqrt( z**2  +  (h/vrms(iz))**2 )
        slope = anti * ( (1./vrms(iz))**2 ) * h / (t+dt)
                                     it  = 1.0 + (t-t0 ) / dt
```

```
        tp = t + abs(slope * dx) + dt;   itp = 1.0 + (tp-t0) / dt
        tm = t - abs(slope * dx) - dt;   itm = 1.0 + (tm-t0) / dt
        if( itm <= 1  )  next
        if( itp >= nt )  break
        amp = 1.0 * sqrt( nt*dt/(t+dt)) * (z+dt)/(t+dt) * (dt/(dt+tp-tm)) ** 2
        xstart = 1-ih;          xstart = max0(  1, xstart)
        xend   = nx-ih;         xend   = min0( nx, xend)
        call spot3x(conj, -amp,2*amp,-amp, t0,dt, tm,t,tp,xstart,xend,
        ih,iz,nt,nx,trmodl(1,iz),trdata(1,itm  ),trdata(1,it  ),trdata(1,itp  ),
                                 trdata(1,itm+1),trdata(1,it+1),trdata(1,itp+1))
                }
        }
if(conj!=0)
        do ix=1,nx
                do it=1,nt
                        modl(it,ix)=trmodl(ix,it)
if(conj==0)
        do ix=1,nx {
                do it=1,nt
                        ss(it)=trdata(ix,it)
                call doubint( 0, add, nt, ss, data(1,ix))
                }
return; end


#  Scaled linear interpolation.
#  Modified for fast triangular weighted anti-aliased migration
#  by Dimitri Bevc 8-14-92
#
subroutine spot3x(conj, scm,sc,scp, t0,dt, tm,t,tp, xstart,xend,ih,iz,
                nt,nx,  val,vecm ,vec, vecp,
                            vecm1,vec1,vecp1)
integer           conj,                             xstart,xend,ih,iz,nt,nx
real                        scm,sc,scp, t0,dt, tm,t,tp
real                  val(nx),vecm(nx) ,vec(nx), vecp(nx)
real                         vecm1(nx),vec1(nx),vecp1(nx)
integer itcm,itc,itcp, ix
real    tcm,tcp,tc,  frm,fr,frp

tcm = (tm-t0) / dt;     itcm = tcm;      frm = tcm - itcm
tc  = (t -t0) / dt;     itc  = tc;       fr  = tc  - itc
tcp = (tp-t0) / dt;     itcp = tcp;      frp = tcp - itcp
if( conj == 0) {
do ix=xstart,xend
         vecm(ix+ih) = vecm(ix+ih)  + (1.-frm) * val(ix) * scm
do ix=xstart,xend
          vec(ix+ih) = vec(ix+ih)   + (1.-fr ) * val(ix) * sc
do ix=xstart,xend
         vecp(ix+ih) = vecp(ix+ih)  + (1.-frp) * val(ix) * scp
do ix=xstart,xend
        vecm1(ix+ih) = vecm1(ix+ih) +  frm     * val(ix) * scm
do ix=xstart,xend
         vec1(ix+ih) = vec1(ix+ih)  +  fr      * val(ix) * sc
do ix=xstart,xend
```

```
        vecp1(ix+ih) = vecp1(ix+ih) +  frp     * val(ix) * scp
        }
else    {
 do ix= xstart, xend
         val(ix) = val(ix)+
                 ((1.-fr )*vec (ix+ih) + fr *vec1 (ix+ih))*sc   +
                 ((1.-frm)*vecm(ix+ih) + frm*vecm1(ix+ih))*scm  +
                 ((1.-frp)*vecp(ix+ih) + frp*vecp1(ix+ih))*scp
        }
return; end
```

TIMING TESTS

Timing results are provided for two small input data sets. The following tables compare the fast anti-aliasing program `kfastaax()` to the tutorial program `kirchfast()` and to `trimo()`.

The following table of cpu times is for a 128×128 input data set run on the SPARCstation 2 OAS:

	`kfastaax()`	`kirchfast()`	`trimo()`
`conj=0`	6.4	1.8	76.8
`conj=1`	5.1	1.8	77.2

The following table of cpu times is for a 256×64 input data set run on the SPARCstation IPC ANTARES:

	`kfastaax()`	`kirchfast()`	`trimo()`
`conj=0`	7.2	2.3	75.0
`conj=1`	6.0	2.4	74.6

`kirchfast()` is about 12 times faster than `trimo()` and only three times slower than `kfastaax()`. The speed-up is especially noticeable on very large data sets.

SYNTHETIC DATA EXAMPLE

The program has been used to generate many figures appearing elsewhere in this report (Black et al., 1992a,b; Bevc, 1992). Figure 1 illustrates the advantage of the anti-aliasing concept.

The synthetic data are generated with the anti-aliasing turned on to simulate field arrays. The image is generated by running `kfastaax()` with `anti=1` and then performing a causal half-order time derivative. The effect of simulating field arrays is that the input impulses are broadened. The point diffractors appear as hyperbolic trajectories with a 45° phase shift on the wavelet.

The migrations are performed with (`anti=1`) and without (`anti=0`) anti-aliasing and are followed by an anticausal half-order time derivative. The aliased migration (lower right of Figure 1) exhibits systematic noise. The point diffractors have upturned artifacts which appear on adjacent traces as part of a smile-like feature. The flat event has a prominent precursor which looks like a layer above the actual event. Anti-aliasing removes the systematic noise and results in a much better image.

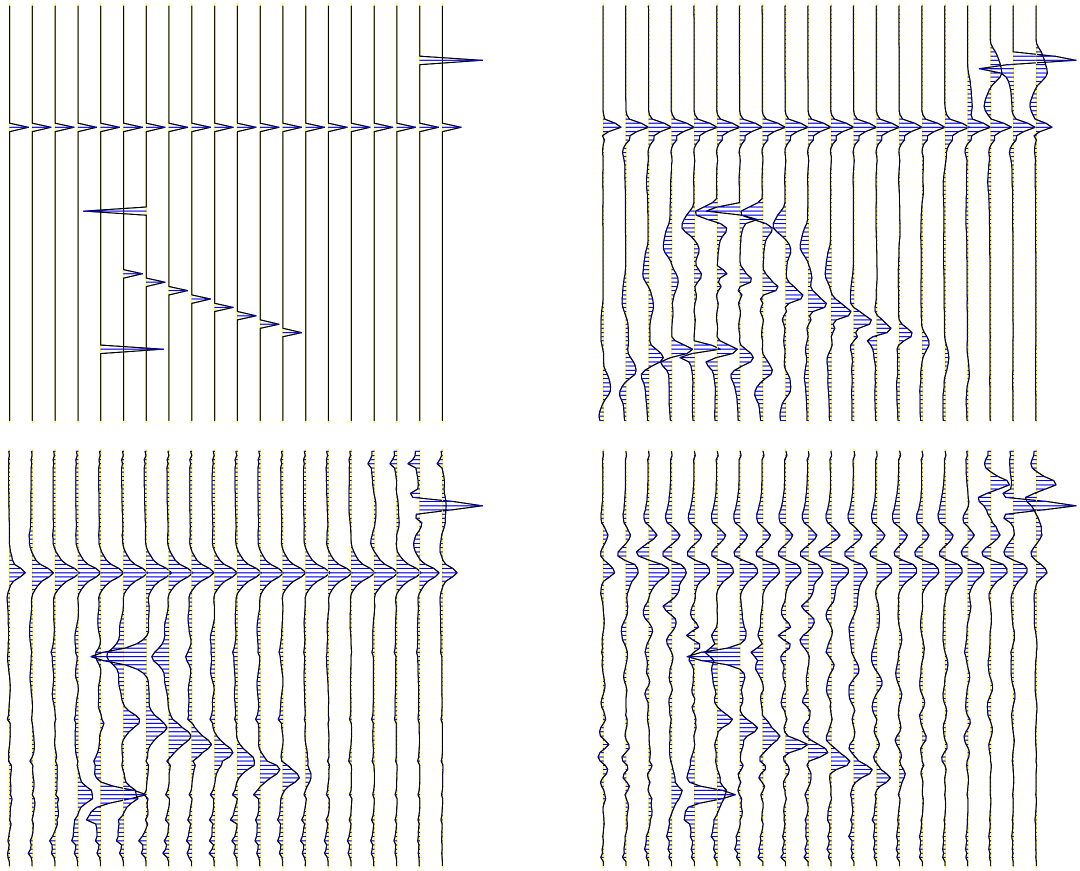

Figure 1: Top left is synthetic input. The top right synthetic data was made by setting the anti-alias parameter `anti=1`. The bottom figures are migrations with and without anti-alias. Bottom left has `anti=1` and bottom right has `anti=0`. dimitri3-migalias [ER]

REFERENCES

Bevc, D., 1992, Kirchhoff wave-equation datuming with irregular acquisition topography: SEP–**75**, 137–156.

Black, J. L., Bevc, D., Brzostowski, M. A., Mo, L.-W., and Palacharla, G., 1992a, Plumes: Response of time migration to lateral velocity variation: SEP–**75**, 21–36.

Black, J. L., Bevc, D., and Palacharla, G., 1992b, Migration velocity dependence of plumes: SEP–**75**, 37–48.

Claerbout, J. F., 1992a, Anti aliasing: SEP–**73**, 371–390.

Claerbout, J. F., 1992b, Introduction to Kirchhoff migration programs: SEP–**73**, 361–366.