

```

1  !S3Main.f95      Group ID: #3      Stereo-3D Simulation Environment Vsn:1.00
2  !2025.05.24.1840cdt- Main program & some utilities.
3
4  !      Author- Jeffrey M. Setterholm, Lakeville,MN 55044 USA
5  !      IP Status- Free source code (e.g.: post copyright)
6  !
7  !      Computer- "T3"/Dell Precision T3500/Intel i5 E5520/win10Pro-21H2
8  !                  ^name ^mfgr.Id      ^chipset      ^OS
9  !                  /Absoft Pro Fortran 21.0.2/GeForce GTX 1050/f90gl~Glut3.7
10 !                  ^compiler ~Fortran 95      ^graphics card ^graphics
11
12 !      f90gl bindings- public domain; see "https://math.nist.gov/f90gl/"
13
14 !Disclaimer:
15 ! *****
16 ! ***** Individual cognition is always flawed, *****
17 ! ***** including yours and mine. *****
18 ! ***** - So: - *****
19 ! ***** Use this code at your own risk. *****
20 ! *****
21
22 !Table of Contents: ...use to search...
23 ! Program      S3Main
24 ! Subroutine BreakHere(iP,Label)
25 ! Subroutine jPause(Label)
26 ! Subroutine SaveOutFile
27 ! Subroutine M44V4multh(V4outh,Matrix44,V4inh,iP)
28 ! Subroutine M44multh(M44outh,M44Linh,M44Rinh,iP)
29 ! Subroutine h44Fill(Hout,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P)
30 ! Subroutine Invertr8(N,MatrixIn,MatrixOut,iP)
31 ! Subroutine Invertr16(N,A,ValMin,iRank,DetN,iUsed,iP)
32 ! Subroutine Printr16(N,A,Noise,iRank,DetN,iRu,jCu,iP)
33 ! Subroutine Rand16(n,Xrandom,iP)
34 ! Subroutine X12Y12toMB(X1,X2,Y1,Y2,M,B,iP)
35 ! Subroutine Eval(Stage,Descr1,Descr2,Value1,Value2,iP)
36 ! Subroutine DaTime18
37 ! Subroutine RunSeconds
38 ! Subroutine PixelDraw                                     = S18AppF8
39 ! Subroutine nRunSec(nClock,RunSecs,iw)
40 ! Subroutine Jdate20(DaTime)
41 !-----7-9
42
43 Program S3Main
44 !2025.01.03.0900cst JMS-
45 !--Globals                                     S1ModDef.f95: 2024.03.18
46 use TaskDef ,only: & !Project/Context
47     ExeName,ExeVersion,ExeBanner
48 use ioDef ,only: & !Files,Units,TimeStamp,Selfies,Flags
49     ExeFileIn,ExeFileOut,EnvNm1,EnvIni,LExists &
50     ,UserIni,UserNm1 &
51     ,Ur,Us,uT,Uread,DaTimeLabel &
52     ,Pi16 ,ModePi ,IterPi
53 use S2Callback,only: GlutHandoff
54 use ViewDef ,only: FrustNom
55 use SimDef ,only: RunSecs
56 use AppsDef ,only: AppNumber,AppNumberNew
57
58 !--End Globals
59 implicit none
60 !--Arguments
61
62 !--Internals

```

```

63  real(8)    ::RunSecsM
64  integer(4)::i
65  !real(16)::valIn, valSqrOut
66
67  !S3Main-CallYourApps.f956/ImportYourNML supercedes use of this:
68  NAMELIST / Sn3Dnml / ExeFileIn, ExeFileOut      &
69                      ,UserIni  , UserNm1        &
70                      ,ExeBanner, ExeName, ExeVersion &
71                      ,Ut        , FrustNom
72
73  !---EndDefs-----
74  call nRunSec(0,RunSecsM,6)
75  ! Modifying key variables at runtime using NameList file `Sn.nml`:
76  ! Initial defaults are set here:
77
78  if(0>1) then !Bypassed-----
79  ! superceded by ImportYourNm1() in S3Main-CallYourApps.f95- called below.
80  Ut      = 13      !Output unit#: Set to 6 or 13
81  EnvNm1  = '..\Sn3D.nml' !default
82  ExeFileOut = '..\Sn3D-Out.txt' ! "
83
84  write(Us, "('S3Main: reading ', :,79a1)") (EnvNm1(i:i), i=1, len_trim(EnvNm1))
85  inquire(file=EnvNm1, exist=LEXists)
86  if(LEXists) then
87      open( unit=Ur, file=EnvNm1, action='read' &
88            , access='sequential', status='old', err=7)
89      read(Ur, nml=Sn3Dnml, err=8) !<- fixed target defined here.
90      close(Ur); goto 10
91  7  pause "Error opening `SnNm1.nml` file. Press enter to continue." ;goto 9
92  8  pause "Error reading `SnNm1.nml` file. Press enter to continue."
93  9  close(Ur)
94  else
95      write(Us,*) "Didn't find `SnNm1.nml`."
96  endif!LEXists
97  10 continue
98  ! write(Us, "('SnNm1.nml: UserNm1(2) =', :,79a1)") &
99  ! (UserNm1(2)(i:i), i=1, len_trim(UserNm1(2))) !Diagnostic
100  endif!(!(0>1) end Bypassed-----
101
102  call ImportYourNm1 !In S3Main-CallYourApps.f95
103
104  write(6, "('S3Main: ExeFileOut = ', :,79a1)") &
105      (ExeFileOut(i:i), i=1, len_trim(ExeFileOut))
106  if(Ut<10) goto 18
107  open(unit=Ut, file=ExeFileOut, action='write', err=19); goto 20
108  18 Ut=Us; write(Us, "(/'You disabled the output text file by setting Ut<10')")
109      write(Us, " ('Ut=Us; output text -> DOS screen.')")
110      goto 20
111  19 Ut=Us; write(Us, &
112      "(/'Unable to open the output text file (e.g. running from a CD)')")
113      write(Us, " ('Ut=Us; & output text -> DOS screen.')")
114  20 call DaTime18
115      write(Ut, "(a18, ' Begin.', 5x, a22, 1x, a22)") DaTimeLabel, ExeName, ExeVersion
116
117  !Call/test a non-visual subroutine here:
118  if(0>1) then !----- Bypasses when (0>1); enters using (1>0)
119      !call
120      call SaveOutFile !...this will save the printout.
121      stop'S3Main @L1117'
122  endif!(0>1) ----- end bypass
123
124  !Perform your application-specific pre-OpenGL initialization(s), if any:
125  ! call

```

```

126 !call pre-OpenGL initializations done.
127 if(AppNumber<1) AppNumber = 1
128 if(AppNumber>4) AppNumber = 1
129
130 call GlutHandoff !OpenGL will take control...
131 End Program S3Main
132 !-----7 9
133
134 Subroutine BreakHere(iP,Label)
135 !2020.05.10.0505cdt JMS- Call this routine from a problem area in your code.
136 ! call BreakHere(iP,'Label') !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
137 ! call BreakHere(iP) !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
138 implicit none
139 !--Arguments
140 integer(4) ::iP !write enable>5: write(iP,...)
141 character(*),intent(in),optional::Label !Ref.automatic: P.109
142 !--Internals
143 integer(4)::i,j
144 !--EndDefs-----
145 if(iP>5) then
146 if(present(Label)) then ;write(iP,"('BreakHere:',\)")
147 j=len_trim(Label); if(j>0) write(iP,"( 80a1:)") (Label(i:i),i=1,j)
148 endif!present: Label
149 endif!iP>5
150 return !<- Breakpoint. From here, you'll return to the calling location.
151 End Subroutine BreakHere
152 !-----7 9
153
154 Subroutine jPause(Label)
155 !2022.04.19.1340cdt JMS- Replaces the deprecated "Pause".
156 implicit none !arguments
157 character(*),intent(in)::Label !Label to write to unit# iWU, "=":no-op
158 !---- !internals
159 integer(4) ::i,j
160 !----- end defs
161 ! Determine the Label length:
162 j = len_trim(Label)
163 ! Write Label + " Press `Enter` to continue: "
164 if(j>0) write(6,"(80a1: )",advance="no") (Label(i:i),i=1,j)
165 write(6,"(' Press Enter` to continue: ')",advance="no")
166 ! The pause is waiting for a keyboard return:
167 read(5,*) !Keyboard attended, hence pausing
168 return
169 end Subroutine jPause
170 !-----7 9
171
172 Subroutine SaveOutFile !2023.00.02.0640
173 !2023.06.24.1155cdt JMS
174 !--Globals
175 use ioDef,only:ExeFileOut,Ut
176 !--End Globals
177 implicit none
178 !--Arguments
179 !--Internals
180 !--EndDefs-----
181 if(Ut<10) return
182 close(Ut)
183 open(Ut,file=ExeFileOut,action='write',position='Append') ;return
184 End Subroutine SaveOutFile
185 !-----7-9
186

```

```

187 Subroutine M44V4multh(V4outh,Matrix44,V4inh,iP)
188 !2020.05.05.1745cdt JMS- V4outh(4) = Matrix44(4,4) * V4inh(4)
189 implicit none
190 !--Arguments
191 real(8) ::V4outh( 4 )
192 real(8) ::Matrix44(4,4)
193 real(8) ::V4inh( 4 )
194 integer(4)::iP !Write enable>5: write(iP,...)
195 !--Internals
196 integer(4)::i,j
197 real(8) ::Vtemp(4)
198 !--EndDefs-----
199 do i=1,4
200     Vtemp(i) = Matrix44(i,1)*V4inh(1) &
201               + Matrix44(i,2)*V4inh(2) &
202               + Matrix44(i,3)*V4inh(3) &
203               + Matrix44(i,4)*V4inh(4)
204 enddo!i
205 Vtemp=Vtemp/Vtemp(4)
206 if(iP>5) then
207     write(iP,"(5x,'V(4)out',27x,'M(4,4)',26x,'V(4)in')")
208     write(iP,"(f12.6,' ',4f12.4,' ',f12.6)") &
209           Vtemp(1),(Matrix44(1,j),j=1,4),V4inh(1)
210     write(iP,"(f12.6,' = ',4f12.4,' * ',f12.6)") &
211           Vtemp(2),(Matrix44(2,j),j=1,4),V4inh(2)
212     write(iP,"(f12.6,' ',4f12.4,' ',f12.6)") &
213           Vtemp(3),(Matrix44(3,j),j=1,4),V4inh(3)
214     write(iP,"(f12.6,' ',4f12.4,' ',f12.6)") &
215           Vtemp(4),(Matrix44(4,j),j=1,4),V4inh(4)
216 endif !iP>5
217
218 V4outh=Vtemp
219 return
220 End Subroutine M44V4multh
221 !-----7 9
222
223 Subroutine M44multh(M44outh,M44Linh,M44Rinh,iP)
224 !2025.05.02.1255cdt JMS- M44outh(4,4) = M44Linh(4,4) * M44Rinh(4)
225 implicit none
226 !--Arguments
227 real(8) ::M44outh(4,4)
228 real(8) ::M44Linh(4,4)
229 real(8) ::M44Rinh(4,4)
230 integer(4)::iP !Write enable>5: write(iP,...)
231 !--Internals
232 integer(4)::i,j
233 real(8) ::M44temp(4,4)
234 !--EndDefs-----
235 do i=1,4
236     do j=1,4
237         M44temp(i,j) = M44Linh(i,1)*M44Rinh(1,j) &
238                       + M44Linh(i,2)*M44Rinh(2,j) &
239                       + M44Linh(i,3)*M44Rinh(3,j) &
240                       + M44Linh(i,4)*M44Rinh(4,j)
241     enddo!j
242 enddo!i
243 if(iP>5) then
244     write(iP,"(4x,'M44outh(4,4)',37x,'= M44Linh(4,4)',33x,'+ M44Rinh(4,4')")
245     do i=1,4
246         write(iP,"(4f12.6,' ',4f12.4,' ',4f12.6)") &
247               (M44temp(i,j),j=1,4),(M44Linh(i,j),j=1,4),(M44Rinh(i,j),j=1,4)
248     enddo!i
249 endif !iP>5

```

```

249      endif !IP>5
250
251      M44outh = M44temp
252      return
253 End Subroutine M44multh
254 !-----7 9
255
256 Subroutine h44Fill(Hout,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P)
257 !2020.04.08.2055cdt JMS- homogeneous(4,4)= A row-order set of 16 coefficients.
258 implicit none      !- fills a matrix which has column-order coefficients.
259 !--Arguments
260 real(8) ::Hout(4,4)      !The 4x4 matrix to fill
261 real(8) ::A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P !The coefficients of Hout
262 integer(4)::iP           ! >5: Printing enable, use: write(iP,...)
263 !--Internals
264 !--EndDefs-----
265 Hout(1,1:4)=(/ A,B,C,D /)
266 Hout(2,1:4)=(/ E,F,G,H /)
267 Hout(3,1:4)=(/ I,J,K,L /)
268 Hout(4,1:4)=(/ M,N,O,P /)      ;return
269 End Subroutine h44Fill
270 !-----7-9
271 Subroutine Invertr8(N,MatrixIn,MatrixOut,iP)      !2020.02.24 JMS
272 !2020.05.01.0550cdt JMS- A(0:15,0:30) <-correction
273 !MatrixOut(N,N) = Inverse[MatrixIn(N,N)] by sequential column processing.
274 !      Homogeneous matrices invert and multiply
275 !      just like other matrices...
276 !      an amazingly useful fact!
277 !"Matrix inversion" is hyperspace's generalization of division.
278 !      If: [Y]=[A]*[X]
279 !      Then: [X]=[Y]/[A] doesn't exist, -but-
280 !      [X]=Inverse[A]*[Y] is defined -but-
281 !      [A] has to be "full rank" to invert, which is like being non-zero in 1D.
282 !Advanced topics:
283 !      >Inverters more powerful than "Invertr8" provide partial inverses
284 !      when matrices are less than "full rank" but not all zero's.
285 !      >"Pseudoinverters" provide "least-squares best fits" of arbitrarily
286 !      large datasets, and very few lines of code can accomplish that.
287 !-----
288 implicit none      !arguments
289 integer(4)::N
290 real(8) ::MatrixIn( N,N)
291 real(8) ::MatrixOut(N,N)
292 integer(4)::iP      !write enable>5: write(iP,...)
293 !---      !internals
294 real(8) ::A(0:15,0:30) !This intermediate array supports 1<= N <=15
295      !The 0th row is swap space.
296 integer(4)::i,iUse,j,Jouter,k
297 real(8) ::Smallest,Largest,Temp,Det
298 !character::Msg*79
299 !character::Msg*50
300 !-----
301 !if(N< 1) then; Msg="sub:Invertr8: N< 1"; call AbortNow(Msg); endif !n< 1
302 !if(N>15) then; Msg="sub:Invertr8: N>15"; call AbortNow(Msg); endif !n>15
303 if(N< 1) stop "sub:Invertr8: N< 1"      !n< 1
304 if(N>15) stop "sub:Invertr8: N>15"      !n>15
305      A      = 0.d0      !Clear the working arrays.
306      A(1:N,1:N)= MatrixIn      !Import the input matrix.
307 do i=1,N; A(i ,i+N)= 1.d0;enddo! !Appends an identity matrix
308
309 !Find the largest abs value:
310 Largest=0.d0

```

```

311 do i=1,N;do j=1,N; if(dabs(A(i,j))>Largest) Largest=dabs(A(i,j));enddo;enddo
312 if(Largest==0.d0)then
313   !Msg="sub:InvertMM: input was all zero's."; call AbortNow(Msg)
314   !Msg="sub:InvertMM: input was all zero's."; stop Msg
315   stop "sub:InvertMM: input was all zero's."
316 endif !Largest==0.
317 !Set the noise floor at a billionth of the largest matrix coefficient:
318 Smallest=Largest/1.d9
319 Det=1.d0
320
321 !Solve
322                                     iUse=-1
323 do Jouter=0,N   !Outer loop:
324   if(Jouter==0) goto 100
325
326   !Find the largest remaining coefficient.
327   Largest=0.d0                                     ; iUse= 0
328   do i=Jouter,N
329     if(dabs(A(i,Jouter))>Largest) then
330       Largest=dabs(A(i,Jouter)); iUse= i
331     endif !A(i,j)>Largest
332   enddo!i
333   if(iUse==0) goto 90 !MatrixIn is less than full rank.
334
335   Largest=A(iUse,Jouter) !The signed value.
336   Det=Det*Largest
337   !Divide the "used row" by the "Largest" value:
338   A(iUse,1:2*N)=A(iUse,1:2*N)/Largest
339
340   !If necessary - swap rows:
341   if(iUse/=Jouter) then
342     A( 0 ,0:2*N)=A( iUse ,0:2*N)
343     A( iUse ,0:2*N)=A(Jouter,0:2*N)
344     A(Jouter,0:2*N)=A( 0 ,0:2*N)
345   endif !iUse/=Jouter
346
347   if(iP>5) then
348     write(iP, &
349       "( 'Process column',i3,', largest was',F12.6,' (' ,i2,',',i2,')')") &
350       Jouter , Largest , iUse ,Jouter
351     write(iP,"(21x,'Det=',f24.12)") Det
352     do i=1,N
353       write(iP,"(30f12.6)") (A(i,j),j=1,2*N)
354     enddo!i
355   endif !iP>5
356
357   !Reduce the Jouter column to an identity.
358   do i=1,N; if(i.eq.Jouter) cycle
359     Temp=A( i ,Jouter)
360     do j=1,2*N; A(i,j)=A(i,j)-Temp*A(Jouter, j ); enddo!j
361   enddo!i
362   goto 100 !Inversion proceeding normally...
363
364 90 continue !Full inversion failed...
365 !Report & exit...
366
367 100 continue
368 if(iP>5) then !Report progress:
369   if(Jouter==0) &
370     write(iP,"(/'sub:Invertr8, N=',i2,' initial setup:',45('-'))") N
371   if(Jouter> 0) write(iP,"( 'Iteration:',i2,' done:')") Jouter
372   do i=1,N

```

```

373      write(iP,"(30f12.6)") (A(i,j),j=1,2*N)
374      enddo!i
375
376      if(Jouter==N) then !Final pass: evaluate the quality of the result.
377      write(iP,"( 'Residual Errors of [MatrixOut]*[MatrixIn]-[I]:',\)")
378      write(iP,"( '      (All zero's would be ideal.)')")
379      do i=1,N
380      do j=1,N; Temp=0.d0; if(i==j) Temp=Temp-1.d0
381      do k=1,N; Temp=Temp+A(i,N+k)*MatrixIn(k,j); enddo!k
382      write(iP,"(d15.6,\)") Temp
383      enddo!j
384      write(iP,*)
385      enddo!i
386      write(iP,"('sub:Invertr8 done.')")
387      endif !Jouter==N
388      endif !iP>5
389
390      if(iUse==0) then
391      !Msg="sub:Invertr8: Failed. MatrixIn wasn't invertable"
392      !call AbortNow(Msg)
393      stop "sub:Invertr8: Failed. MatrixIn wasn't invertable"
394      endif !n< 1
395      !Inversion proceeding
396      enddo!Jouter
397
398      MatrixOut=A(1:N,N+1:2*N)
399      return
400 End Subroutine Invertr8
401 !-----7 9
402
403 Subroutine Invertr16(N,A,ValMin,iRank,DetN,iUsed,iP)
404 !2010.01.06.1205cst JMS- Traveler2/Athlon64/WinXPro-32/APF9.0-32
405 !2006.05.25.0715cdt JMS- An instantiation of the "Setterholm Matrix Inverter"
406 !                               Expects,uses, & returns real(16)'s (~QUAD precision).
407 !-----No globals
408 implicit none!arguments
409 integer(4):: N
410 real(16) :: A(N,N)
411 real(16) :: ValMin
412 integer(4):: iRank
413 real(16) :: DetN !Abs(determinant product) for each iteration
414 integer(4):: iUsed(N)
415 integer(4):: iP
416 !-----!internals
417 integer(4),allocatable::iRu(:)
418 integer(4),allocatable::jCu(:)
419 real(16),allocatable::Temp(:)
420 integer(4)::i,iu,iu2,j,ju,ju2,L
421 real(16) ::Amult,ValMax
422 integer(8)::iBinSum
423 integer(4)::iAlloc
424 !-----!end defs
425 if(iP.gt.5) write(iP,"('Overwriting Matrix Inversion:')")
426 DetN=0._16 ;iUsed(1:N)=0 ;iRank=0
427 !-----
428 allocate(iRu(N),stat=iAlloc)
429 if(iAlloc.ne.0) stop 'Invert: iRu(N) allocation error. Halt.'
430 iRu=0
431 allocate(jCu(N),stat=iAlloc)
432 if(iAlloc.ne.0) stop 'Invert: jCu(N) allocation error. Halt.'
433 jCu=0
434 allocate(Temp(N),stat=iAlloc)

```

```

435  if(iAlloc.ne.0) stop 'Invert: Temp(N) allocation error. Halt.'
436  Temp=0._16
437  !---
438  Temp=0._16; iRu  =0 ;jCu  =0 ;ValMin=0._16
439  do i=1,N ;iRu(i)=i ;jCu(i)=i ;end do
440  if(iP.gt.5) call Printr16(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*****
441  ! Inversion Loop:
442  do L=1,N ;iU=0 ;jU=0 ;ValMax =0._16
443  do i=1,N ;if(iRu(i).lt.0) cycle
444  do j=1,N ;if(jCu(j).lt.0) cycle
445  if(ValMax.ge.abs(A(i,j))) cycle
446  ValMax=abs(A(i,j)) ;iu=i ;ju=j
447  enddo!j
448  enddo!i
449  if(L==1) ValMin=ValMax*1.d-24 !Establishes the "noise floor". was 1.d-20
450  if(L==1) DetN=ValMax
451  if(iU.eq.0) Exit
452  if(ValMax.lt.ValMin) Exit
453  iRank=iRank+1
454  if(L>1) DetN=DetN*ValMax
455  ValMax=A(iu,ju) ;iu2=jCu(ju) ;ju2=iRu(iu)
456  !if(iP.gt.5) write(iP,"(3i3,f15.6)") L,iu,ju,ValMax
457  if(iP.gt.5) write(iP,"(1x,2i4,f15.9,' :iu ju pivot, & value')") &
458  iu,ju, ValMax
459  Temp(1:N)=A(1:N,ju) ;i=iRu(iu)
460  A(1:N,ju)=A(1:N,ju2) ; iRu(iu)=iRu(iu2)
461  A(1:N,ju2)=Temp(1:N) ; iRu(iu2)=-abs(i)
462  Temp(1:N)=A(iu,1:N) ;j=jCu(ju)
463  A(iu,1:N)=A(iu2,1:N) ; jCu(ju)=jCu(ju2)
464  A(iu2,1:N)=Temp(1:N)/ValMax ; jCu(ju2)=-abs(j)
465  do i=1,N ;if(i.eq.iu2) cycle ;Amult=A(i,ju2)
466  do j=1,N ;A(i,j)=A(i,j)-A(iu2,j)*Amult ;enddo!j
467  A(i,ju2)=-Amult/ValMax
468  enddo; A(iu2,ju2)=1._16/ValMax
469  if(iP.gt.5) call Printr16(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*****
470  enddo!L Inversion loop- done.
471
472  iBinSum=0
473  do i=1,n
474  if((iRu(i).lt.0).and.(i.lt.63)) iBinSum=iBinSum+2**(i-1)
475  if(iRu(i).lt.0) iUsed(i)=1
476  ! Zero linearly-dependent rows and columns, if any:
477  if(iRu(i).gt.0) A(i,1:N)=0._16
478  if(jCu(i).gt.0) A(1:N,i)=0._16
479  enddo!i
480  if((iP >5).and.(iu == 0)) then
481  write(iP,"('Linear dependency dealt with. Partial inverse results:')")
482  write(iP,"(b63.2,' -Kp unused flag')") iBinSum !For the first 62 Kp's
483  call Printr16(N,A,ValMin,iRank,DetN,iRu,jCu,iP) !*****
484  endif!iP>5 & iu==0
485  if(iP.gt.5) write(iP,"('Overwriting Inverter- done.')")
486  deallocate(iRu)
487  deallocate(jCu)
488  deallocate(Temp)
489  return
490 End Subroutine Invertr16
491 !-----7-9
492
493 Subroutine Printr16(N,A,Noise,iRank,DetN,iRu,jCu,iP)
494 !2018.09.04.1340cdt JMS- Prints Invert's progress.
495 ! Expects,uses, & returns real(16)'s (~QUAD precision).
496 !--- No globals

```



```

497 implicit none
498 integer(4):: N
499 real(16) :: A(N,N)
500 real(16) :: Noise
501 integer(4):: iRank
502 real(16) :: DetN
503 integer(4):: iRu(N)
504 integer(4):: jCu(N)
505 integer(4):: iP
506 !---
507 integer(4)::i,j
508 !-----
509
510 write(iP,"('Inverter: @ Rank =',i5,8x,'abs(Det)='e39.30/2x,\)") iRank,DetN
511 do j=1,N; write(iP,"(i18,\)") jCu(j) ; enddo; write(iP,"('')")
512 do i=1,N; write(iP,"(i5,' ' ,\)") iRu(i)
513 do j=1,N; write(iP,"(sp,f18.9,\)") A(i,j); enddo; write(iP,"('')")
514 enddo!i
515 write(iP,"('')");
516 End Subroutine Printr16
517 !-----7-9
518
519 Subroutine Rand16(n,Xrandom,iP)
520 !2015.01.31.1645cst JMS- Random number generator
521 ! - Set Xrandom=0._16 prior to first call.
522 !-- Use:-----
523 ! integer*4::nRandom=12
524 ! real*16 ::Xrandom(12)
525 !Xrandom(1)=-1408701324 -or-
526 !Xrandom(1)=0._16
527 !do i=1,10
528 ! call Rand16(nRandom,Xrandom,iP)
529 !enddo!i-----
530 implicit none
531 !--Arguments
532 integer(4)::n
533 real(16) ::Xrandom(n)
534 integer(4)::iP
535 !--Internals
536 integer(4)::Init
537 integer(4)::nRand !Number of integers in the seed
538 integer(4)::iSeed(4)
539 integer(4) time
540 !--EndDefs-----
541
542 !-- Random number generation initialization:
543 if(Init.eq.0) then
544 call random_seed
545 call random_seed(size=nRand) != 1
546 call random_seed(get=iSeed(1:nRand)) != -1408701324 (default value)
547 if(iP.gt.5) write(iP,"('Rand16: nRand=',i2,' iSeed=',10(i14:))") &
548 nRand , iSeed(1:nRand)
549 iSeed=0
550 !-- Initialize randomness...
551 if(Xrandom(1).eq.0._16) then
552 !-- ... using the computer's seconds clock when Xrandom(1)=0.:
553 iSeed(1)=time() !e.g.: = 1422733593 @ 2015.01.31.1346.33
554 ! = 1422733725 @ 2015.01.31.1348.46
555 !Returns the seconds since 00:00:00 GMT January 1. 1970
556 else
557 !-- ... by you setting the integer value(s) of Xrandom(1:nRand):
558 iSeed(1:nRand)=Xrandom(1:nRand)+sign(.1_16,Xrandom(1:nRand))

```

```

559     endif !
560
561     call random_seed(put=iSeed(1:nRand))
562     if(iP.gt.5) write(iP,"(5x,'Value used: iSeed=',10(i14:))") iSeed(1:nRand)
563     Init=1
564     endif !Xrandom(1)=0.
565
566     call random_number(Xrandom)
567 ! For iSeed(1)=-1408701324 (default value)
568 ! & n=2:
569 ! 1 0.1390874385833740234375000 0.6339781284332275390625000
570 ! 2 0.2369287014007568359375000 0.4304550290107727050781250
571 ! & n=1:
572 ! 1 0.1390874385833740234375000
573 ! 2 0.6339781284332275390625000 < ^ :Exactly the same sequence.
574 ! 3 0.2369287014007568359375000
575 ! 4 0.4304550290107727050781250
576     if(iP.gt.5) then
577         if(Init.le.999) then; write(iP,"('Random# set:',i3,')") Init
578         else; write(iP,"('R#:',i12,')") Init
579         endif!Init<=999
580         if(n.lt.3) then; write(iP,"( 2(f26.22:) )") Xrandom(1:n)
581         else; write(iP,"(667( /3(f26.22:) ) )") Xrandom(1:n)
582 !Rand16: nRand= 1 iSeed= -1408701324
583 ! Value used: iSeed= 1422743635
584 !Random# set: 1
585 ! 0.6806479096412658691406 0.6724007725715637207031 0.0493627786636352539063
586 ! 0.4378559589385986328125 0.2765567302703857421875 0.4999549388885498046875
587 ! 0.3916140198707580566406 0.3895425200462341308594 0.3124939203262329101563
588 ! 0.6459279060363769531250 0.5968738198280334472656 0.4783441424369812011719
589 !Random# set: 2
590 ! 0.7516865134239196777344 0.2383043169975280761719 0.4446819424629211425781
591 ! 0.7403323650360107421875 0.0192565321922302246094 0.0300652384757995605469
592 ! 0.5761010050773620605469 0.7211880683898925781250 0.7397087216377258300781
593 ! 0.9435601830482482910156 0.7619655728340148925781 0.2017260193824768066406
594     endif !n<3
595     endif !iP>5
596     Init=Init+1
597     return
598 End Subroutine Rand16
599 !-----7-9
600
601 Subroutine X12Y12toMB(X1,X2,Y1,Y2,M,B,iP)
602 !2020.05.13.0840cdt JMS- Slope & intercept defined by a line's endpoints.
603 implicit none
604 !--Arguments
605 real(8) ::X1,X2,Y1,Y2 !Line endpoints
606 real(8) ::M,B !Slope & Intercept Y=M*X+B
607 integer(4):: iP !Write enable>5: write(iP,...)
608 !--Internals
609 !--EndDefs-----
610 M=(Y2-Y1)/(X2-X1); B=Y1-M*X1 ;if(iP<6) return
611 !--Print results to unit iP
612 write(iP,"(/'X12Y12toMB: X2=',f15.6,' Y2=',f15.6)") X2,Y2
613 write(iP,"( ' X1=',f15.6,' Y1=',f15.6)") X1,Y1
614 write(iP,"( ' Deltas: ',f15.6,' ',f15.6)") (X2-X1),(Y2-Y1)
615 write(iP,"( ' Hence: M =',f15.6,' B =',f15.6)") M,B ;return
616 End Subroutine X12Y12toMB
617 !-----7-9
618
619 Subroutine Eval(Stage,Descr1,Descr2,Value1,Value2,iP)
620 !2020.05.07.1345cdt JMS- Incremental numerical derivation confirmation.

```

```

621 implicit none
622 !--Arguments
623 character(*),intent(in)::Stage,Descr1,Descr2                !Ref.automatic: P.109
624 real(8) ::Value1,Value2
625 integer(4)::iP                      ! >5: Printing enable, use: write(iP,...)
626 !--Internals
627 integer(4)::i
628 !--EndDefs-----
629 if(iP<6) return
630 write(iP,"('/Eval Stage: ',60a1)") (Stage(i:i),i=1,len_trim(Stage))
631 write(iP,"(f19.9,' = ',60a1: )") Value1,(Descr1(i:i),i=1,len_trim(Descr1))
632 write(iP,"(f19.9,' = ',60a1: )") Value2,(Descr2(i:i),i=1,len_trim(Descr2))
633 return
634 End Subroutine Eval !Usage example:
635 !--
636 ! call Eval( "#1->#5" &                                60 char limit-> " &
637 !           , "-( (b+1)*(2*d+m)+4*d-(b-1)*(m-2*d) )" &
638 !           , "-(4*b*d+4*d+2*m)" &
639 !           , "-( (b+1)*(2*d+m)+4*d-(b-1)*(m-2*d) )" &
640 !           , "-(4*b*d+4*d+2*m)" ,iP)
641 !Eval Stage: #1->#5
642 !      -236.210494995 = -( (b+1)*(2*d+m)+4*d-(b-1)*(m-2*d) )
643 !      -236.210494995 = -(4*b*d+4*d+2*m)!Eval Stage: #1->#5
644 !-----7-9
645
646 Subroutine DaTime18
647 !2020.04.05.1510cdt JMS- Present date & time string -> DaTimeLabel*18
648 !--Globals                                S1ModDef.f95: 2024.03.18
649 use ioDef,only:DaTimeLabel !Input/Output & Flags
650 !--End Globals
651 implicit none
652 !--Internals
653 integer(4)::i,iDMY(3),iHMS(3)
654 !--EndDefs-----
655
656 call iDate(iDMY)
657 call iTime(iHMS)
658 write(DaTimeLabel,"( i4,'.',i2.2,'.',i2.2,'.',i2.2,i2.2,'.',i2.2)") &
659 (iDMY(i),i=3,1,-1),(iHMS(i),i=1,3) ;return
660 End Subroutine DaTime18
661 !-----7 9
662
663 Subroutine RunSeconds
664 !2020.06.16.0640cdt JMS- Run seconds.                                Not verified.
665 !--Globals
666 use SimDef,only:RunSecs !Input/Output & Flags
667 !--End Globals
668 implicit none
669 !--Internals
670 integer(4)::Init=0
671 integer(4)::Vo(8),V(8)
672 integer(4)::V3prev,iDays=0
673 ! integer(4)::DayWrap=0
674 !--EndDefs-----
675
676 if(Init==0) then
677   call date_and_Time(VALUE=Vo)
678   V=Vo
679   Init=1
680 endif!Init=0
681 V3Prev=V(3)
682 call date_and_Time(VALUE=V)

```

```

683   if(V(3)/=V3Prev) iDays=iDays+1
684   RunSecs = ((iDays*24+V(5)-Vo(5))* 60.d0 & !Hours
685             +V(6)-Vo(6))* 60.d0 & !Minutes
686             +V(7)-Vo(7) & !Seconds
687             +(V(8)-Vo(8))/1000.d0 !Millisec
688   return
689 End Subroutine RunSeconds
690 !-----7 9
691
692 !This is example app F8:
693
694 Subroutine PixelDraw !2020.02.04.0740cst JMS
695 !Draws magnifiable pixels directly onto the OpenGL screen.
696 ! Support seeing pixel-level algorithms running live step-by-step.
697 !use OglCanMod !<- S.*, OpenGL, & the environment's subroutines ! globals
698 !-----
699
700 use OpenGLRec !Ref: OpenGL GL/GLU/GLUT docs
701 use ioDef ,only: Up !Files,Units,TimeStamp,Selfies,Flags
702 use ScreenDef, only: xwindowFull,ywindowFull,ColorRGBA
703 use AppsDef ,only: AppBanner,AppInFile
704 use ViewDef ,only: ThreePhase !View bounds->Unit Cube ->[extent]Destination
705 !use OglCanDef !, only: S
706 !use OglCanMod, only: Colors3D,PrntOrtho
707 implicit none !arguments
708 !-- !internals
709 integer(4)::MtxMode(1)
710 real(8) ::Orth6(6)
711 integer(4) ::i
712 real( kind=glFloat)::w0,h0, PixMagf
713 real( kind=glFloat)::fImage(3)=(/ 0., 1., 1. /)
714 real(8) ::PixMag=32
715 integer(4)::iColor
716 character(len=80)::PText
717 !----- ! end defs
718 AppBanner(8) = 'PixelDraw 2020.02.04.1800' !a58
719 AppInFile(8) = char(0) !a35
720 if(Up>5)write(Up, "('F8 PixelDraw: ThreePhase=',i2,' Up=',i2)")ThreePhase,Up
721 select case(ThreePhase)
722 case(1) !Background number crunching, if any:
723 case(2) !Update variables & 2D screen info: Superceded by split screen(s)
724 case(3) !Draw/redraw app. 2D & 3D graphics:
725
726 call glGetIntegerv(GL_MATRIX_MODE,MtxMode)
727 call glMatrixMode(GL_MODELVIEW) ;call glPushMatrix ;call glLoadIdentity
728 call glMatrixMode(GL_PROJECTION) ;call glPushMatrix ;call glLoadIdentity
729
730 call glMatrixMode(GL_PROJECTION) ;call glLoadIdentity
731 call glMatrixMode(GL_MODELVIEW) ;call glLoadIdentity
732
733 iColor=0
734 Orth6=(/ 0.d0 , xwindowFull/PixMag &
735 , ywindowFull/PixMag, 0.d0 &
736 , -1.d0 , 1.d0 /)
737 call glOrtho( Orth6(1),Orth6(2),Orth6(3),Orth6(4),Orth6(5),Orth6(6) )
738
739 !Alter screen pixel (w0,h0):
740 PixMagf=PixMag
741 call glPixelZoom(PixMagf,PixMagf)
742 do i=1,495/idint(PixMag)
743 iColor=iColor+1; if(iColor.gt.16) iColor=1
744 call Colors3D(iColor); FImage=ColorRGBA(1:3)

```

```

745
746             ! w0=i; h0=i*2.0
747             w0=0; h0=i
748     call glRasterPos2f( w0 , h0 )
749     call glDrawPixels(1_GLsizei,1_GLsizei,GL_RGB,GL_FLOAT,fImage)
750
751     write(Ptext,"('Color#:',i3)") iColor
752             call PrntOrtho(iColor+4,50,iColor, 0,PText)
753
754     enddo!i
755     call glFlush
756
757 16 call glMatrixMode(GL_MODELVIEW ) ;call glPopMatrix
758     call glMatrixMode(GL_PROJECTION) ;call glPopMatrix
759     call glMatrixMode(MtxMode(1))
760
761     end select!ThreePhase
762     return
763 End Subroutine PixelDraw
764 !-----7 9
765
766 Subroutine nRunSec(nClock,RunSecs,iw)
767 !2023.08.12.1645cdt JMS- Run seconds for (0:30) separate clocks
768 !--Globals ...1. millisecond resolution
769 use ioDef, only: DaT,DaTo
770 !--End Globals
771 implicit none
772 !--Arguments
773 integer(4) ::nClock !<0:reset&start; >0:read; =0:time since first call
774 real(8) ::RunSecs !nClock`s elapsed seconds
775 integer(4) ::iw
776 !--Internals
777 integer(4) ::Init = 0
778 integer(4) ::nClockL
779 character ::DaTime*20
780 !--EndDefs-----
781 call Jdate20(DaTime) !Computes DaT
782 nClockL = nClock
783 if(Init==0) then
784     DaTo(0) = DaT
785     Init = 1
786 endif!(Init==0)
787 select case(nClock)
788 case( :-31); Stop 'nRunSec(nClock <-30... undefined. Halt@L125'
789 case(-30:- 1); nClockL = -nClockL; DaTo(nClockL) = DaT
790 case(+31: ); Stop 'nRunSec(nClock >+30... undefined. Halt@L127'
791 end select!(nClock)
792 RunSecs = (DaT%Sec -DaTo(nClockL)%Sec ) &
793           +(DaT%JDay-DaTo(nClockL)%JDay)*86400.d0
794 !if(iw>5) write(iw,"( 5x,'RunSecs(',i2,')=',f27.3)") nClock,RunSecs
795 if(iw>5) write(iw,"(10x,SP,i3,S,f31.3)") nClock,RunSecs
796
797                                     return
798 End Subroutine nRunSec
799 !-----7 9
800 Subroutine Jdate20(DaTime) !2023.08.20.1210
801 !2023.08.11.0600cdt JMS- Reads a snapshot of the computers current date & time.
802 use ioDef, only:DaT
803 implicit none
804 !--Arguments
805 character::DaTime*20
806 !--Internals

```

```

807 integer(4)::V(8)
808 integer(4)::iYear,iMonth,iDay,iHour,iMin,iSec,imSec
809 !integer(4)::JulianDay
810 !real(8) ::DaySecond
811 !--EndDefs-----
812 call date_and_Time(VALUE=V) !V(4) =Time offset w.r.t. UTC
813 iYear = V(1); iHour = V(5)
814 iMonth = V(2); iMin = V(6)
815 iDay = V(3); iSec = V(7)
816 imSec = V(8) !includes: seconds
817 write(DaTime,"(i4,'.',i2.2,'.',i2.2,'.',i2.2,i2.2,'.',i2.2,' L' )") &
818 iYear,iMonth,iDay,iHour,iMin,iSec
819 write(DaT%cDaT22, & !includes: milliseconds
820 "(i4,'.',i2.2,'.',i2.2,'.',i2.2,i2.2,'.',i2.2,'.',i3.3)") &
821 iYear,iMonth,iDay,iHour,iMin,iSec,imSec
822 !JulianDay
823 DaT%jDay = 367* iyear &
824 -7* (iyear+(imonth+9)/12) /4 &
825 -3*((iyear+(imonth-9)/ 7)/100+1)/4 &
826 +275* imonth /9+iday+1721029
827 !DaySecond
828 DaT%Sec = (iHour*60.d0+iMin)*60.d0+iSec*1.d0+imSec/1000.d0
829 !Use DaySecond (& JulianDay) for differential timing to 1 millisec:
830 return
831 End Subroutine Jdate20
832 !-----7 9
833

```