```
function produce_los, files, dataall

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; If given, data needs to have x, y, z, dx, dy, dz or the corners
;;
;; common block contains
;;   * input
;;   * format
;;   * SystemConsts
;;   * stuff = {aplanet, vrplanet, atoms_per_packet, mod_rate, totalsource}
;;   * gvalue = {lines, velocity, g}
;;   * plasma = TBD
;;
;; Version History:
;;   4.11: 12/8/2011
;;     * Need to make sure it doesn't use too many packets at once
;;   4.8: 7/20/2011
;;     * adding ability to use cylinder instead of instrument FOV
;;     * adding more comments
;;     * possible bug fixes
;;   4.6: 4/21/2011
;;     * Makes use of parallelized kd_tree code
;;   4.5: 4/20/2011
;;     * same as 4.6 with debugging info still included.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common constants
common results

;; Determine which mechamisms to do
doresscat = (max(strcmp(format.emission.mechanism, 'resscat', /fold)))
doeimp = (max(strcmp(format.emission.mechanism, 'eimp', /fold)))

;; Determine points and lines of sights
geometry = format.geometry
geotags = strlowcase(tag_names(geometry))

;; Determine how dr is set
formtags = strlowcase(tag_names(geometry))
q = fix(total(strmatch(tag_names(format), 'dr', /fold)))
if (q) $
  then dr = format.dr $
  else dr = geometry.dr

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Load the data if not given
if (size(dataall, /type) NE 8) then begin
  ;; figure out what information is given
  tag_spacecraft = total(stregex(geotags, 'spacecraft', /bool))
  tag_orbit = total(stregex(geotags, 'orbit', /bool))
  tag_phase = total(stregex(geotags, 'phase', /bool))
  tag_tstart = total(stregex(geotags, 'tstart', /bool))
  tag_tend = total(stregex(geotags, 'tend', /bool))
  tag_dt = total(stregex(geotags, 'dt', /bool))

  if (tag_spacecraft EQ 0) then begin
    print, 'A spacecraft must be specified for LOS measurements.'
    stop
  endif
  sc = strlowcase(geometry.spacecraft)
  case (sc) of
    'messenger': begin
      ;; can specify either (tstart, tend) or (orbit, phase)
      ;; Orbit only currently makes sense for the flybys
      case (1) of
        (tag_orbit): begin
          phase =  (tag_phase) ? geometry.phase : 'all'
          dataall = load_MASCS_data(input.options.atom, geometry.orbit, phase, /Level3, $
            /model)
          end
        (tag_tstart) and (tag_tend): $
          dataall = load_MASCS_data(input.options.atom, geometry.tstart, geometry.tend, $
            /Level3, /model)
        else: begin
          print, 'Not set up yet.'
          stop
          endelse
      endcase
      end
    else: stop
  endcase
  if (strcmp(dataall.species, 'none', /fold)) then stop
endif
;; Now have data = {x, y, z, xbore, ybore, zbore, xcorner, ycorner, zcorner},

sss = (where(strlowcase(*SystemConsts.Objects) EQ $
  strlowcase(input.geometry.StartPoint)))[0]
robj = (sss EQ 0) ? SystemConsts.rplan*1e5 : $    ;; radius of object in cm
  SystemConsts.rplan*(*SystemConsts.radius)[s]*1e5

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determine which observations are too far from the planet and which
;; look at the planet
;; Distance of s/c from planet
dist_from_plan = sqrt(*dataall.x^2 + *dataall.y^2 + *dataall.z^2)

;; Angle between look dir and planet -- negative since want from look pt to planet
ang = acos((-*dataall.x**dataall.xbore - *dataall.y**dataall.ybore - $
  *dataall.z**dataall.zbore)/dist_from_plan)

```

```
;; Remove observations not looking close enough to the object
if ~(input.options.fullsystem) then begin
  mindist = dist_from_plan * sin(ang)
  todo = (mindist LE input.options.OuterEdge)
endif else todo = replicate(1, n_elements(*dataall.x))

data = data_extract(dataall, todo)

;; check to see if look direction intersects the planet anywhere
;; angular size of planet from look pt.
asize_plan = asin(1./dist_from_plan)

;; Don't worry about lines of sight that don't hit the planet
missp = where(ang GT asize_plan, nmissp, comp=hitp)
if (nmissp GT 0) then dist_from_plan[missp] = 1e30


t0 = systime(1)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Now look at the model outputs
nf = n_elements(files)
nspec = n_elements(*data.x)
nall = n_elements(*dataall.x)

if (dr EQ 0) then begin
  ;; Use Voronoi method
  stop

  rhosqr_sun = *out.x^2 + *out.z^2
  out_of_shadow = ((rhosqr_sun GT 1) or (*out.y LT 0))
  ;; construct the voronoi regions and a kdtree to determine the los density
  print, 'Using instrument FOV'

  ;; make the voronoi region for these points
  regions = results_voronoi(out2)

  ;; make the kd_tree for these points
  tree = results_kd_tree(out2)

  ;; Determine FOV
  phic = atan(*data.ycorner, *data.xcorner)   ;; Corners
  thc = asin(*data.zcorner) & sinthc = *data.zcorner
  phib = atan(*data.ybore, *data.xbore) & thb = asin(*data.zbore) ;; Boresight

  ;; Determine where LOS intersects modeled region
  limits = results_find_intersection_points(data, input)
  lim0 = reform(limits[0,*]) & lim1 = reform(limits[1,*])
  m0 = min(phic, dim=1) & m1 = max(phic, dim=1)
  l0 = min(sinthc, dim=1) & l1 = max(sinthc, dim=1)

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; Loop over each individual LOS
  nr = 1000L & nphi = 15L & nth = 15L

  radiance = dblarr(nspec)
  density = dblarr(nr,nspec)
  denr = dblarr(nr,nspec)
  rrr = dindgen(nr)/(nr-1) & ppp = dindgen(nphi)/(nphi-1) & ttt = dindgen(nth)/(nth-1)
  iii = one(ppp) & jjj = one(ttt)

  tstart = systime(1)
  for i=0,nspec-1 do begin
    if (todo[i]) then begin
      t0 = systime(1)
      rtemp = rrr*(lim1[i]-lim0[i]) + lim0[i]
      ddr = rtemp[1]-rtemp[0]
      phitemp = ((ppp*(m1[i]-m0[i]) + m0[i]) # jjj)[*]
      sinthtemp = (iii # (ttt*(l1[i]-l0[i]) + l0[i]))[*]
      thtemp = asin(sinthtemp)

      xtemp0 = cos(phitemp)*cos(thtemp)
      ytemp0 = sin(phitemp)*cos(thtemp)
      ztemp0 = sinthtemp

      roi = obj_new('IDLanROI', phic[*,i], sinthc[*,i])
      q = where(roi.ContainsPoints(phitemp, sinthtemp), nq)
      obj_destroy, roi

      xden = (xtemp0[q]#rtemp)[*] + (*data.x)[i]
      yden = (ytemp0[q]#rtemp)[*] + (*data.y)[i]
      zden = (ztemp0[q]#rtemp)[*] + (*data.z)[i]

      den1 = results_density(xden, yden, zden, out2, regions, tree)
      den1 = reform(den1, nq, nr)
      denr[*,i] = rtemp
      density[*,i] = total(den1, 1)/nq
      radiance[i] = total(density[*,i])*ddr*robj
      t1 = systime(1)
      print, 'LOS Spec Number: ' + strint(i+1) + ' of ' + strint(nspec), t1-t0
    endif
  endfor
  tend = systime(1)
  print, 'LOS time: ', tend-tstart

;; Determine slit solid angle
;;omega = slit_solidangle(data)

endif else begin
  ;; Use a uniform thickness cylinder to determine los column density
  ;; Area of a column
  Apix = !pi * (dr*robj)^2

```

```
;; Load the packets
xx = !null & yy = !null & zz = !null & frac = !null & radvel_sun = !null
for ff=0,nf-1 do begin
  results_loadfile, files[ff], pts, vels_sun, frac2 ;; note - not keeing frac=0
  xx = [xx, pts[*,0]] & yy = [yy, pts[*,1]] & zz = [zz, pts[*,2]]
  frac = [frac, frac2]
  radvel_sun = [radvel_sun, vels_sun[*,1]+stuff.vrplanet]  ;; for g-value
  print, 'Loaded inputs ' + strint(ff+1) + ' of ' + strint(nf)
endfor
out = {x:ptr_new(temporary(xx)), y:ptr_new(temporary(yy)), $
  z:ptr_new(temporary(zz)), frac:ptr_new(temporary(frac)), $
  radvel_sun:ptr_new(temporary(radvel_sun))}

;; Determine emission measure for each packet
;; base shadow on whether los goes through shadow
out_of_shadow = replicate(1, n_elements(*out.x))
weight = results_packet_weighting(out, out_of_shadow)

radiance = fltarr(nspec)
for i=0,nspec-1 do begin
  ;; Determine which packets are close to the line of sight
  xpr = *out.x - (*data.x)[i]
  ypr = *out.y - (*data.y)[i]
  zpr = *out.z - (*data.z)[i]
  rpr = sqrt(xpr^2 + ypr^2 + zpr^2)
  costheta = (xpr*(*data.xbore)[i] + ypr*(*data.ybore)[i] + $
    zpr*(*data.zbore)[i])/rpr

  ;; delta = perpendicular distance to the line of sight
  delta = rpr * sin(acos(costheta))
  q = where(finite(delta) EQ 0, nq) & if (nq GT 0) then stop

  inview = where((delta LT dr) and (costheta GT 0) and (*out.frac GT 0), nin)

  if (nin GT 0) then begin
    ftemp = (*out.frac)[inview]/apix
    if (doresscat) then begin
      ;; Determine whether the point along the LOS the packet represents is in
      ;; shadow
      losr = rpr[inview] * costheta[inview] ;; projection of packet onto LOS
      xhit = (*data.x)[i] + (*data.xbore)[i]*losr ;; point packet represents
      yhit = (*data.y)[i] + (*data.ybore)[i]*losr
      zhit = (*data.z)[i] + (*data.zbore)[i]*losr
      rhohit = xhit^2 + zhit^2
      out_of_shadow = (rhohit GT 1) or (yhit LT 0)
      ftemp *= out_of_shadow
    endif

    radiance[i] = total(ftemp)
  endif
  if ((i mod 100) EQ 0) then print, 'Finished spec #' + strint(i)
  endfor
  result = {radiance:ptr_new(radiance), format:format}
endelse

return, result

end
```