```
function produce_voronoi_image, files

common constants
common results

;;;;;;;;;;;;;;;;
;; Determine the image origin
s = (where(strcmp(*SystemConsts.objects, format.geometry.origin, /fold), ns))[0]
if (ns NE 1) then stop

;;;;;;;;;;;;;;;;
;; Determine image field of view and rotation
geometry = format.geometry

image = dblarr((geometry.dims)[0],(geometry.dims)[1])
immin = geometry.center - geometry.width/2.
immax = geometry.center + geometry.width/2.

scale = geometry.width/(geometry.dims-1)        ;; [xscale,zscale] in Rplan/pix
Apix = (scale[0]*scale[1])*(SystemConsts.rplan*1e5)^2 ;; cm^2/pix

;; xaxis and zaxis in Robj measured from center of object
xaxis = findgen((geometry.dims)[0])*scale[0] + immin[0]
zaxis = findgen((geometry.dims)[1])*scale[1] + immin[1]

;; Determine frame rotation
M = determine_image_rotation(input, format)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
xx = !null & yy = !null & zz = !null & frac = !null
vx = !null & vy = !null & vz = !null & radvel_sun = !null
nf = n_elements(files)
for ff=0,nf-1 do begin
  ;; restore output file and extract useful packets
  ;; pts_sun is in solar reference frame with origin=Object center, units R_obj
  ;; vels_sun in km/s
  results_loadfile, files[ff], pts_sun, vels_sun, frac2 ;; note - not keeing frac=0

  ;; Rotate the packets to observer frame
  pts_obs = M ## pts_sun    ;; observer along -y axis
  vels_obs = M ## vels_sun

  ;; Determine which packets are not blocked by the planet
  rhosqr_obs = pts_obs[*,0]^2 + pts_obs[*,2]^2  ;; rho in observer's frame
  inview = ((rhosqr_obs GT 1) or (pts_obs[*,1] LT 0))
  frac2 *= inview

  h = where((pts_obs[*,0] GE immin[0]) and (pts_obs[*,0] LE immax[0]) and $
    (pts_obs[*,2] GE immin[1]) and (pts_obs[*,2] LE immax[1]), nh)
  if (nh GT 0) then begin
    xx = [xx, pts_obs[h,0]] & yy = [yy, pts_obs[h,1]] & zz = [zz, pts_obs[h,2]]
```

```
52      vx = [vx, vels_obs[h,0]] & vy = [vy, vels_obs[h,1]] & vz = [vz, vels_obs[h,2]]
53      frac = [frac, frac2[h]]
54      radvel_sun = [radvel_sun, vels_sun[h,1]+stuff.vrplanet]  ;; for g-value
55    endif
56    print, 'Loaded inputs ' + strint(ff+1) + ' of ' + strint(nf)
57  endfor
58  out = {x:ptr_new(temporary(xx)), y:ptr_new(temporary(yy)), $
59    z:ptr_new(temporary(zz)), frac:ptr_new(temporary(frac)), $
60    vx:ptr_new(temporary(vx)), vy:ptr_new(temporary(vy)), $
61    vz:ptr_new(temporary(vz)), radvel_sun:ptr_new(temporary(radvel_sun))}
62
63  weight = results_packet_weighting(out, format)
64  ;; Additional factors:
65  case (format.quantity) of
66    'column': weight /= Apix
67    'intensity': weight /= Apix
68    'density': weight /= Vpix
69    else: stop
70  endcase
71
72  ;; Determine voronoi regions
73  regions = results_voronoi(out)
74
75  ;; make the kd_tree for these points
76  tree = results_kd_tree(out)
77
78  dy = min(scale)/2.
79  ny = round((max(*out.y)-min(*out.y))/dy)+1
80  yaxis = findgen(ny)*dy + min(*out.y)
81
82  density = dblarr((geometry.dims)[0],(geometry.dims)[1],ny)
83  yy = (one(zaxis)#yaxis)[*]
84  zz = (zaxis#one(yaxis))[*]
85  nn = n_elements(yy)
86  for i=0,(geometry.dims)[0]-1 do begin
87    t0 = systime(1)
88    xx = replicate(xaxis[i],nn)
89    temp = results_density(xx, yy, zz, out, regions, tree)
90    density[i,*,*] = reform(temp, (geometry.dims)[1], ny)
91    t1 = systime(1)
92    print, i, t1-t0
93  endfor
94
95  ;;for i=0,(geometry.dims)[0]-1 do begin
96  ;;  for j=0,(geometry.dims)[1]-1 do begin
97  ;;    density[i,j,*] = results_density(replicate(xaxis[i],ny), yaxis, $
98  ;;      replicate(zaxis[i],ny), out, regions, tree)
99  ;;  endfor
100 ;;  print, i
101 ;;endfor
102
```

```
103  result = {image:ptr_new(image), xaxis:ptr_new(xaxis), zaxis:ptr_new(zaxis), $
104    format:format, yaxis:ptr_new(yaxis), density:ptr_new(density)}
105  return, result
106
107  end
```