```
pro driver, input, output, seed=seed

;;**************************************************************************
;;
;;  Driver routine to run the 5th order RK integrator from Numerical
;;  Recipies, 3rd Ed.
;;
;;  Version History:
;;   3.1: 4/27/2011
;;     -- Need to speed up when modeling satellites
;;   3.0: 7/20/2010
;;     -- Revising for new structure architecture
;;   2.7: 7/6/2010:
;;     -- Adding impact_check_2.9 to this program so it does not use the include
;;   2.6: 4/26/2010:
;;     -- Added moon's temperature map
;;     -- removed thermalized option from emitfn case (no longer used)
;;   2.5: 1/14/2010:
;;     -- Keep track of fate of each packet
;;     -- Keep track of deposition on the surface
;;     -- Replace ptr_free with destory_structure
;;   2.4: 12/7/2009
;;     -- Allowing variable surface temperature for Maxwellian reemission
;;   2.3: 11/6/2009
;;     -- changing the way it does the thermalization. New velocity is determined from
;;         partial accomodation to thermal speed at surface
;;   2.2: Added variable surface temperature for particle sticking (for Mercury,
;;     Based on surface temperature in Leblanc & Johnson 2003). -- I don't think I
;;     did this [12/7/09]
;;   2.1: Added support for elastic bouncing of particles from the surface
;;   2.0: Revised for current structure setup
;;   1.2: Previous working version
;;   1.1: Older version to work with rk4
;;   1.0: Similar to the original version to work with rk7
;;
;; Impact_check version history (before inclusion into this program):
;;   Version 2.9 4/28/10
;;      -- Changing definition of accommodation coefficient
;;         * Need energy accommodation rather than velocity accommodation
;;         * Before: v_1 = a v_th + (1-a) v_0
;;         * After: v_1^2 = a v_th^2 + (1-a) v_0^2
;;   Version 2.8 3/9/10
;;      -- Fixing issue with thermal accomodation - Now choose speed based on
;;     thermal distribution.
;;   Version 2.7 1/19/10
;;      -- fixing some problems with v2.6
;;   Version 2.6 1/14/2010
;;      -- keep track of what happens to each packet
;;      -- keep track of surface deposition
;;           * doesn't get the map right for satellites.
;;   Version 2.4 11/6/2009
;;      -- added thermal accomodation to the surface
;;   Version 2.1 -- added option for elastic bouncing
;;
;;***************************************************************

common constants

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Remake the loc structure to speed up the math and make it easier to read
if (input.options.trackloss) $
  then loc = {t:ptr_new(0), x:ptr_new(0), v:ptr_new(0), frac:ptr_new(0), $
    lossfrac:ptr_new(0), hitfrac:ptr_new(0), ringfrac:ptr_new(0)} $
  else loc = {t:ptr_new(0), x:ptr_new(0), v:ptr_new(0), frac:ptr_new(0)}
*loc.x = [[*output.x], [*output.y], [*output.z]]
*loc.v = [[*output.vx], [*output.vy], [*output.vz]]
*loc.frac = *output.frac
*loc.t = *output.time ;; This is how much time before present and works up to zero
npack = n_elements(*output.x)

if (input.options.trackloss) then begin
  lossfrac = dblarr(npack)
  hitfrac = dblarr(npack,n_elements(*SystemConsts.objects))
  ringfrac = dblarr(npack)
  leftfrac = dblarr(npack)
  deposition = {longitude:ptr_new(findgen(360)*!dtor), $
    latitude:ptr_new(findgen(180)*!dtor-!pi/2), $
    map:ptr_new(dblarr(360,180,n_elements(*SystemConsts.objects)))}
  ilon = findgen(360)
  ilat = findgen(180)
endif

which = where(*input.geometry.include, nw)
pp = replicate(1., n_elements(*SystemConsts.objects))

;; Set up the stepsizes
h = replicate(1000d, npack)     ;initial guess at best stepsize
hold = h  ;; last step used by each packet

;Set variables in preparation for iteration
count = 0L  ;; number of steps taken

;These control how quickly the stepsize is increased or decreased between iterations
safety = .95
shrink = -.25
grow = -.2

;; yscale = scaling parameter for each variable
;;   x,y,z ~ R_plan
;;   vx,vy,vz ~ 1 km/s (1/Rplan Rplan/s)
;;   frac ~ exp(-t/lifetime) ~ mean(frac)

```

```
resolution = input.options.resolution

;; Set up the bounce conditions
if (input.sticking_info.stickcoef NE 1.) then begin
  case strlowcase(input.sticking_info.emitfn) of
    'maxwellian': begin
      if (input.sticking_info.Tsurf EQ 0) then begin   ;; Use surf temp model
        case (input.geometry.startpoint) of
          'Mercury': begin
            temp0 = 100.
            temp1 = 600 + 125*(cos(input.geometry.taa)-1)/2. ;; sub-solar temp fn of taa
            nn = .25
            end
          'Moon': begin
            temp0 = 151
            temp1 = 162.
            nn = .25
            end
          else: stop
        endcase

        nt = 21 & nv = 101
        temperature = dindgen(nt)/(nt-1)*temp1 + temp0
        v_temp = sqrt(2*temperature*!const.kb/atomicmass(input.options.atom)) /1e5
        prob = dindgen(101)/100.
        vgrid = dblarr(nt,101)
        for i=0,nt-1 do begin
          vrange = dindgen(nv)/(nv-1)*v_temp[i]*3.
          f_v = MaxwellianDist(vrange, temperature[i], input.options.atom)
          sumdist = f_v
          for j=1,nv-1 do sumdist[j] += sumdist[j-1]
          sumdist /= max(sumdist)
          vgrid[i,*] = interpol(vrange, sumdist, prob)
        endfor
      endif else begin   ;; use constant surf temp
        v_th = sqrt(2*input.Sticking_info.Tsurf*!const.kb/atomicmass(input.options.atom)) /1e5
        vrange = findgen(1001)/1000 * v_th*5 & vrange = vrange[1:*]
        f_v = MaxwellianDist(vrange, input.Sticking_info.Tsurf, options.atom)

        sumdist = f_v
        for i=1,n_elements(vrange)-1 do sumdist[i] += sumdist[i-1]
        sumdist /= max(sumdist)
      endelse
      end
    'elastic scattering':
    else: stop
  endcase
end

;******************************************************************************
;Keep takeing R.K. steps until every packet has reached the time of "image taken"
```

```
;*******************************************************************************

moretogo = where((*loc.t GT resolution) and (*loc.frac GT 0), ntogo)
done = (ntogo EQ 0)

while ~(done) do begin
  ;Now generate sub-arrays containing only the particles that are still being tracked

  loc0 = {t: ptr_new((*loc.t)[moretogo]), x:ptr_new((*loc.x)[moretogo,*]), $
    v:ptr_new((*loc.v)[moretogo,*]), frac:ptr_new((*loc.frac)[moretogo])}

  w = where(*loc0.frac EQ 0, nw) & if (nw NE 0) then stop
  w = where(finite(*loc0.x) EQ 0, nw) & if (nw NE 0) then stop
  w = where(finite(*loc0.v) EQ 0, nw) & if (nw NE 0) then stop

  oldx = *loc0.x  ;; This is used for determining if anything hit the rings
  oldf = *loc0.frac
  h = hold[moretogo]

  ;Adjust stepsize to be no more than time remaining
  h = (h LE (*loc0.t))*h + (h GT (*loc0.t))*(*loc0.t)

  ;; Run the rk5 step
  rk_5, loc0, h, input, which, delta

  ;; Do the error check
  ;; scale = a_tol + |y| * r_tol
  ;;   for x: a_tol = r_tol = resolution
  ;;   for v: a_tol = r_tol = resoltuon/10. -- require v to be more precise
  ;;   for f: a_tol = 0.01 ; r_tol = 0 -- set fractional tolerance to 1%
  scalespace = resolution + abs(*loc0.x) * resolution
  scalevel = 0.1*(resolution + abs(*loc0.v) * resolution)
  scaleabund = 0.01 ;; resolution + abs(*loc0.frac) * resolution

  ;; difference relative to acceptable difference
  *delta.x /= scalespace
  *delta.v /= scalevel
  *delta.frac /= scaleabund
  xerrmax = max(*delta.x, dim=2)
  verrmax = max(*delta.v, dim=2)

  ;; Maximum error for each packet
  errmax = (xerrmax GE verrmax)*xerrmax + (xerrmax LT verrmax)*verrmax
;  errmax = (errmax GE *delta.frac)*errmax + (errmax LT *delta.frac)**delta.frac
  if ((where(finite(errmax) EQ 0))[0] NE -1) then stop

  ;; Check where difference is very small - adjust step size
  noerr = where(errmax LE 1e-7)
  if (noerr[0] NE -1) then begin
    errmax[noerr] = 1.
    h[noerr] = h[noerr]*10.
  endif

  ;; Put the post-step values
  g = where(errmax LE 1.0, ng, comp=b)
  if (ng GT 0) then begin
    (*loc.t)[moretogo[g]] = (*loc0.t)[g]
    (*loc.x)[moretogo[g],*] = (*loc0.x)[g,*]
    (*loc.v)[moretogo[g],*] = (*loc0.v)[g,*]
    (*loc.frac)[moretogo[g]] = (*loc0.frac)[g]
    if (input.options.trackloss) then $
      lossfrac[moretogo[g]] += (oldf[g]-(*loc0.frac)[g])  ;; add in change in frac
    h[g] = safety*h[g]*errmax[g]^grow
  endif

  if (ng NE ntogo) then begin
    ;; don't adjust the bad values, but do fix the stepsize
    htemp = safety * h[b] * errmax[b]^shrink
    q = where(htemp LT 0.0, nq) & if (nq NE 0) then stop

    ;; don't let step size drop below 1/10th previous step size
    h[b] = max([[htemp], [0.1*h[b]]], dim=2)
  endif
  qqq = where(h LT 1e-7, nqq) & if (nqq NE 0) then stop  ;; error test

  destroy_structure, loc0
  destroy_structure, delta

  ;save new values of h
  hold[moretogo] = h

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;; Impact check
  ;; Only look at packets which moved during this step
  if (ng GT 0) then begin
    ;; Make a new structure with just the packets that moved this step
    loc1 = {t:ptr_new((*loc.t)[moretogo[g]]), x:ptr_new((*loc.x)[moretogo[g],*]), $
      v:ptr_new((*loc.v)[moretogo[g],*]), frac:ptr_new((*loc.frac)[moretogo[g],*])}
    if (input.options.trackloss) then begin
      oldfrac = *loc1.frac
      hitfrac1 = hitfrac[moretogo[g],*]
      ringfrac1 = ringfrac[moretogo[g]]
      leftfrac1 = leftfrac[moretogo[g]]
    endif

    jj = replicate(1., ng)

    ;; 1) Did the packets hit anything?
    ;Get object positions
    if (input.options.motion) $
      then locmoon, *loc1.t, *input.geometry.phi, *SystemConsts.a, $
        *SystemConsts.orbrate, x=xSat, y=ySat, z=zSat, ang=ang $
        else locmoon, fltarr(ng), *input.geometry.phi, *SystemConsts.a, $
          *SystemConsts.orbrate, x=xSat, y=ySat, z=zsat, ang=ang

      ;; Distance of packets from each object
      tempR = sqrt(((*loc1.x)[*,0]#pp - xSat)^2 + ((*loc1.x)[*,1]#pp - ySat)^2 + $
        ((*loc1.x)[*,2]#pp - zSat)^2)

      ;Is r < satellite radius?
      eps = 0.
      satrad = jj # (*SystemConsts.radius)*(1-eps)
      hhh = where((tempR-satrad) LT 0, nhits)
      if (nhits NE 0) then begin
        ;;w = where((*loc1.t)[hhh mod ng] EQ 0, nw) & if (nw NE 0) then stop
        hx = hhh mod ng & hy = hhh/ng

        ;; adject the frac values
        if (input.sticking_info.stickcoef EQ 1) $
          then (*loc1.frac)[hx] = 0 $
          else (*loc1.frac)[hx] = (*loc1.frac)[hx] * (1.-input.sticking_info.stickcoef)

        ;; If need to know where things hit the surface, do this
        if ((input.options.trackloss) or (input.sticking_info.stickcoef LT 1)) then begin
          ;; Figure out where things hit the surface
          srad = satrad[hhh]
          r0 = tempR[hhh]  ;; R_plan
          x0 = (*loc1.x)[hx,0]  ;; R_plan
          y0 = (*loc1.x)[hx,1]  ;; R_plan
          z0 = (*loc1.x)[hx,2]  ;; R_plan
          r0 = sqrt(x0^2 + y0^2 + z0^2)

          ;; Position of the satellites
          xcent = xSat[hx,hy]
          ycent = ySat[hx,hy]
          zcent = zSat[hx,hy]

          ;; Vector from center of satellite to packet
          ;;   -- packet positions relative to satellite
          x1 = (x0-xcent)/srad  ;; rsat
          y1 = (y0-ycent)/srad  ;; rsat
          z1 = (z0-zcent)/srad  ;; rsat

          ;; Velocity - orbital vel = vel relative to satellite
          vxsat = -(*SystemConsts.orbvel)[hy]*cos(ang[hx,hy])*input.options.motion/$
            SystemConsts.rplan
          vysat = -(*SystemConsts.orbvel)[hy]*sin(ang[hx,hy])*input.options.motion/$
            SystemConsts.rplan

          vx0 = (*loc1.v)[hx,0] - vxsat  ;; rplan/s
          vy0 = (*loc1.v)[hx,1] - vysat  ;; rplan/s
          vz0 = (*loc1.v)[hx,2]  ;; rplan/s

      ;; Find where the packet hit the surface
      ;;    |x + vt| = 1 -- see ResearchNotes from 4/28/08
      a = vx0^2 + vy0^2 + vz0^2
      b = 2*(x1*vx0 + y1*vy0 + z1*vz0)
      c = x1^2 + y1^2 + z1^2 - 1

      dd = b^2 - 4*a*c
      q = where(dd LT 0, nq) & if (nq NE 0) then stop
      t0 = (-b - sqrt(b^2-4*a*c))/(2*a)
      t1 = (-b + sqrt(b^2-4*a*c))/(2*a)
      t = (t0 LE 0)*t0 + (t1 LT 0)*t1

      ;; Point where packet hit the surface
      x2 = x1 + vx0*t
      y2 = y1 + vy0*t
      z2 = z1 + vz0*t
      ;; r2 = sqrt(x2^2 + y2^2 + z2^2)  ;; -- this should be = 1.

      lonhit = (atan(x2, -y2) + 2*!pi) mod (2*!pi)
      lathit = asin(z2)

      ;; Put new coordinates into the array
      x_final = xcent + x2*srad
      y_final = ycent + y2*srad
      z_final = zcent + z2*srad

      q = where(finite(x_final) EQ 0, nq) & if (nq NE 0) then stop
      q = where(finite(y_final) EQ 0, nq) & if (nq NE 0) then stop
      q = where(finite(z_final) EQ 0, nq) & if (nq NE 0) then stop
      (*loc1.x)[hx,0] = x_final
      (*loc1.x)[hx,1] = y_final
      (*loc1.x)[hx,2] = z_final
    endif

    if (input.options.trackloss) then begin
      lonind = (fix(interpol(ilon, *deposition.longitude, lonhit))) mod $
        n_elements(*deposition.longitude)
      latind = fix(interpol(ilat, *deposition.latitude, lathit)) mod $
        n_elements(*deposition.latitude)
      for i=0L,nhits-1 do (*deposition.map)[lonind[i],latind[i],hy[i]] += $
        oldfrac[hx[i]]*input.sticking_info.stickcoef
      hitfrac1[hhh] += oldfrac[hx]
    endif

    if (input.sticking_info.stickcoef LT 1) then begin
      ;; Figure out rebound velocity
      vv02 = vx0^2 + vy0^2 + vz0^2  ;; rplan/s
      PE = 2*(*SystemConsts.GM)[hy]*(1./r0-1./srad)
      vv02 += PE
      q = where(vv02 LT 0, nq) & if (nq NE 0) then vv02[q] = 0.
      q = where(finite(vv02) EQ 0, nq) & if (nq NE 0) then stop
```

```
case strlowcase(input.sticking_info.emitfn) of
  'maxwellian': begin ;; Re-emit the packets with a thermal distribution
    if (input.sticking_info.Tsurf EQ 0) then begin
      surftemp = temp0 + (temp1*(abs(cos(lonhit)*cos(lathit)))^nn)*$
        (abs(lonhit) LT !pi/2)
      rr = random_nr(nhits, seed=seed, routine=0)
      vv_new = interpolate_xy(vgrid, temperature, prob, surftemp, rr)/$
        SystemConsts.rplan
    endif else vv_new = interpol(vrange, sumdist, $
      random_nr(seed=seed, nhits))/SystemConsts.rplan ;; rplan/s

    vv2 = sqrt(input.Sticking_info.accom_factor*vv_new^2 + $
      (1-input.Sticking_info.accom_factor)*vv02)
    end
  'elastic scattering': vv2 = sqrt(vv02)
endcase

;; Determine new direction with F(v) ~ cos(theta)
alt = acos(random_nr(seed=seed, nhits))
az = 2*!pi * random_nr(seed=seed, nhits)

v_rad = sin(alt)          ;; Radial component of velocity
v_east = -cos(alt) * sin(az) ;; Component along latitude line (points east)
v_north = cos(alt) * cos(az) ;; Component along longitude line (points to NP)

vx2 = v_rad*x2
vy2 = v_rad*y2
vz2 = v_rad*z2

lat = asin(z2)
lon = atan(x2, y2)

vx2 = dblarr(nhits) & vy2 = dblarr(nhits) & vz2 = dblarr(nhits)
for i=0L,nhits-1 do begin
  M = transpose([ $
    [cos(lat[i])*sin(lon[i]), cos(lat[i])*cos(lon[i]), sin(lat[i])], $
    [-cos(lon[i]), sin(lon[i]), 0], $
    [-sin(lat[i])*sin(lon[i]), -sin(lat[i])*cos(lon[i]), cos(lat[i])] ])
  v_ren = [v_rad[i], v_east[i], v_north[i]]
  v_xyz = invert(M) # v_ren
  vx2[i] = v_xyz[0] * vv2[i]
  vy2[i] = v_xyz[1] * vv2[i]
  vz2[i] = v_xyz[2] * vv2[i]
endfor

;; The new position in planet-centered coords
vx_final = vx2 + vxsat
vy_final = vy2 + vysat
vz_final = vz2
```

```
        q = where(finite(vx_final) EQ 0, nq) & if (nq NE 0) then stop
        q = where(finite(vy_final) EQ 0, nq) & if (nq NE 0) then stop
        q = where(finite(vz_final) EQ 0, nq) & if (nq NE 0) then stop

        (*loc1.v)[hx,0] = vx_final
        (*loc1.v)[hx,1] = vy_final
        (*loc1.v)[hx,2] = vz_final
       endif
      endif

      ;; 2) Have the packets left the corona? (only check if not tracking the full system)
      if ~(input.options.fullSystem) then begin
       leftCor = where(tempR[*,stuff.s] GT input.options.OuterEdge * $
        (*SystemConsts.radius)[stuff.s], hh)
       if (hh NE 0) then begin
        if (input.options.trackloss) then leftfrac1[leftcor] += (*loc1.frac)[leftcor]
        (*loc1.frac)[leftCor] = 0
       endif
      endif

      ;; 3) If Saturn, check to see if anything hit the rings
      if (input.geometry.planet EQ 'Saturn') then begin
       ;; Ring region within 2.3 Rs
       ox = oldx[g,*]
       cross = ox[*,2] * (*loc1.x)[*,2]  ;; if cross is negative, then crossed eq. plane
       MayHit = where(cross LE 0 , nmay)
       if (nmay NE 0) then begin
        orho = sqrt(total(ox[MayHit,0:1]^2, 2))
        nrho = sqrt(total((*loc1.x)[MayHit,0:1]^2, 2))
        w = where((orho LT 2.3) or (nrho LT 2.3), nw)
        for j=0,nw-1 do begin $
         crosspt = interpol([orho[w[j]],nrho[w[j]]], [ox[MayHit[w[j]],2], $
          (*loc1.x)[MayHit[w[j]],2]], 0.)
         if (crosspt LT 2.3) then begin
          if (input.options.trackloss) then ringfrac1[MayHit[w[j]]] += $
           (*loc1.frac)[MayHit[w[j]]]
          (*loc1.frac)[MayHit[w[j]]] = 0.
         endif
        endfor
       endif
      endif

      ;; Check to see if any packets have shrunk out of existence
      q = where((*loc1.frac GT 0) and (*loc1.frac LT 1e-10), nq)
      if (nq NE 0) then (*loc1.frac)[q] = 0.

      ;; If any new hits, set the time remaining to 0.
      w = where(*loc1.frac EQ 0, nw)
      if (nw NE 0) then (*loc1.t)[w] = 0.

      ;Put new values back into original array (again)
      (*loc.t)[moretogo[g]] = *loc1.t
      (*loc.x)[moretogo[g],*] = *loc1.x
      (*loc.v)[moretogo[g],*] = *loc1.v
      (*loc.frac)[moretogo[g]] = *loc1.frac
      if (input.options.trackloss) then begin
        hitfrac[moretogo[g],*] = hitfrac1
        ringfrac[moretogo[g]] = ringfrac1
        leftfrac[moretogo[g]] = leftfrac1
      endif

      destroy_structure, loc1
    endif
    ;;end impact check
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    moretogo = where(*loc.t GT resolution, ntogo)  ;; check to see which ones aren't done
    if (count mod 100 EQ 0) then print, stuff.strstart + 'Step Number: ' + string(count) + $
      ', Packets Remaining: ' + string(ntogo)
    count += 1 ;; step counter

    ;If it goes 100000 steps then it will never stop!
    done = ((ntogo EQ 0) or (count GT 100000.))
  endwhile

  *output.x = reform((*loc.x)[*,0])
  *output.y = reform((*loc.x)[*,1])
  *output.z = reform((*loc.x)[*,2])
  *output.vx = reform((*loc.v)[*,0])
  *output.vy = reform((*loc.v)[*,1])
  *output.vz = reform((*loc.v)[*,2])
  *output.frac = *loc.frac
  if (input.options.trackloss) then begin
    *output.hitfrac = reform(hitfrac)
    *output.lossfrac = lossfrac
    *output.ringfrac = ringfrac
    *output.leftfrac = leftfrac
    output.deposition = deposition
  endif

  destroy_structure, loc

  end

```