Subject: Re: fast search
Posted by greg michael on Thu, 19 Oct 2006 14:24:21 GMT
View Forum Message <> Reply to Message

erm... I mean here...

Greg

```
pro splitsearch3,p,dist
 ;recursively splits the search volume into n_split^3 subvolumes. When
 ;there are fewer than 'threshold' points
 ;in a subvolume, checks for matches the brute force way - every point
 ;against every other.

 n_split=4        ;1-D cutting factor (for 3, cube is cut into 3x3x3=27
subvolumes)
 threshold=75     ;no. of points to start pairwise comparison
 n=n_elements(p)  ;no. of points in cloud

 if n gt threshold then begin ;if more than threshold points, further
divide the volume
  mxx=max(p.x,min=mnx) ;get range of x
  mxy=max(p.y,min=mny) ;get range of y
  mxz=max(p.z,min=mnz) ;get range of z

  ;determine which subrange of x,y,z each point belongs to:
  bx=fix((p.x-mnx)/(mxx-mnx)*n_split)<(n_split-1) ;< to ensure max
element not in new bin
  by=fix((p.y-mny)/(mxy-mny)*n_split)<(n_split-1)
  bz=fix((p.z-mnz)/(mxz-mnz)*n_split)<(n_split-1)

  ;determine which subvolume each point belongs to (for n_split=3 there
will be 3x3x3=27)
  b=bx+by*n_split+bz*n_split^2

  ;get reverse_indices - a ordered list telling which points lie in
which subvolume (bin)
  ;                    Note that how this works is obscure - check
here to understand:
  ;
http://www.dfanning.com/tips/histogram_tutorial.html
  h=histogram(b,min=0,reverse_indices=ri)

  for i=0,n_elements(h)-1 do begin
   if ri[i] ne ri[i+1] then begin ;see again histogram tutorial
    q=[ri[ri[i]:ri[i+1]-1]]       ;see again histogram tutorial

    ;if there are more than two points in the subvolume, call this
```

```
routine with just those points
    ;this is what makes the routine recursive. Eventually there will be
fewer than threshold points,
    ;and then the second half of the routine gets executed, with the
brute force comparison.
    if n_elements(q) ge 2 then splitsearch3,p[q],dist ;splitsearch
again, if enough to compare
   endif
  endfor

 endif else begin
  ;This is the brute force comparison. Using q1 and q2 as indices to p
lets you compare every element
  ;against every other. d is a matrix of these pairwise distances
  q1=rebin(indgen(n),n,n) ;set up indices for pairwise matching
  q2=transpose(q1)
   d=sqrt((p[q1].x-p[q2].x)^2+(p[q1].y-p[q2].y)^2+(p[q1].z-p[q2 ].z)^2)
;calculate pair distances
  i=where((d le dist) and (q1 gt q2)) ;select close neigbours (q1>q2 to
avoid duplicate pairs)

  ;this is where the results come out:
  if i[0] ne -1 then print,p[q1[i]],p[q2[i]] ;print the neighbours, if
found
 endelse
end
```