Subject: Re: fast search
Posted by Paolo Grigis on Tue, 17 Oct 2006 12:53:11 GMT
View Forum Message <> Reply to Message

This link may be useful to you:

http://www.dfanning.com/code_tips/pts_in_sphere.html

Ciao,
Paolo

m.goullant@gmail.com wrote:
> Hi there,
>
> I have the following problem:
>
> ;data structure of an irregular point cloud
> x = points.x
> y = points.y
> z = points.z
>
> search radio
> radio = 8
>
> FOR i=0L,N_ELEMENTS(z)-1 DO BEGIN
>
> square = WHERE(x LE x[i] + radio AND x GE x[i] - radio AND y LE
> y[i] + radio AND y GE y[i] - radio)
> ;(...)
>
> ENDFOR
>
> I realize that WHERE will do the job, but at very low efficiency.
> WHERE
> makes no assumptions about the list being ordered. It seems to me it
> has
> to check every element of the array, requiring N steps for an N-element
> array
>
> There is a faster way to do this?
>
> thanks,
> Marie
>

Subject: Re: fast search
Posted by greg michael on Tue, 17 Oct 2006 14:15:12 GMT
View Forum Message <> Reply to Message

Hi Marie,

Also take a look at this thread...

http://groups.google.de/group/comp.lang.idl-pvwave/browse_th
read/thread/1d020842b165598a/8e57c4b2c3841851?#8e57c4b2c3841 851

A program to search for pairs closer than a given distance d in a point cloud. It recursively cuts up the volume until there are fewer than a given threshold of points in the subvolume, and then directly compares those few. At each cutting of the space, it's necessary to leave an overlap of d so that points in adjoining subvolumes aren't missed. I don't remember - looks like it might be necessary to check for the uniqueness of the solutions at the end. Anyway, the recursion is the fun part...

By the way, it looks like there are a lot of for-loops there, but they're powerful ones - each run of the loop cuts the space.

regards,
Greg

m.goullant@gmail.com wrote:
> Hi there,
>
> I have the following problem:
>
> ;data structure of an irregular point cloud
> x = points.x
> y = points.y
> z = points.z
>
> search radio
> radio = 8
>
> FOR i=0L,N_ELEMENTS(z)-1 DO BEGIN
>
> square = WHERE(x LE x[i] + radio AND x GE x[i] - radio AND y LE
> y[i] + radio AND y GE y[i] - radio)
> ;(...)
>
> ENDFOR
>
> I realize that WHERE will do the job, but at very low efficiency.
> WHERE

> makes no assumptions about the list being ordered. It seems to me it
> has
> to check every element of the array, requiring N steps for an N-element
> array
>
> There is a faster way to do this?
>
> thanks,
> Marie

Subject: Re: fast search
Posted by greg michael on Tue, 17 Oct 2006 17:17:57 GMT
View Forum Message <> Reply to Message

Actually, there's a mistake in that code - it's got mixed
procedure/function syntax. Should be like this:

```
function ss_MakeGalaxies,n
 p=replicate({x:0.,y:0.,z:0.},n)
 seed=0
 p.x=randomu(seed,n)*1000.
 p.y=randomu(seed,n)*1000.
 p.z=randomu(seed,n)*1000.
 return,p
end

function ss_distance,p1,p2
 return,sqrt((p2.x-p1.x)^2+(p2.y-p1.y)^2+(p2.z-p1.z)^2)
end

pro splitsearch,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=3
 threshold=75

 s=(size(p))[1]

 if s gt threshold then begin
  xmn=min(p.x,max=xmx)
  ymn=min(p.y,max=ymx)
  zmn=min(p.z,max=zmx)

  xg=xmn+findgen(n_split+1)*(xmx-xmn)/n_split ;grid boundaries
```

```
  yg=ymn+findgen(n_split+1)*(ymx-ymn)/n_split
  zg=zmn+findgen(n_split+1)*(zmx-zmn)/n_split

  for j=0,n_split-1 do begin
   for k=0,n_split-1 do begin
    for l=0,n_split-1 do begin
     w= where(  (p.x ge xg[j]-dist) and (p.x lt xg[j+1]+dist) and $
               (p.y ge yg[k]-dist) and (p.y lt yg[k+1]+dist) and $
                     (p.z ge zg[l]-dist) and (p.z lt zg[l+1]+dist))
     if n_elements(w) ge 2 then begin
      splitsearch,p[w],dist
     endif
    endfor
   endfor
  endfor

 endif else begin
  for i=0,s-2 do begin
   for j=i+1,s-1 do begin
    if ss_distance(p[i],p[j]) le dist then begin
     print,p[i],p[j]
    endif
   endfor
  endfor
 endelse
end
```

Then call it using:

```
IDL> p=ss_MakeGalaxies(1e5)
IDL> splitsearch,p,.5
{     454.665      175.794      87.0097}{     454.816      175.504
  87.3471}
{     556.761      981.076      933.809}{     556.654      980.922
  934.065}
{     981.340      196.286      105.551}{     981.387      196.102
  105.703}
```

The left and right columns are the neigbouring (< .5 units distant) xyz triples.

regards,
Greg

Subject: Re: fast search
Posted by greg michael on Tue, 17 Oct 2006 18:25:39 GMT

And here's a proper IDL version that uses histograms and the pairwise comparisons that David likes... still got three loops, though - have to think some more to get rid of those.

```
function ss_MakeGalaxies,n
 p=replicate({x:0.,y:0.,z:0.},n)
 seed=0
 p.x=randomu(seed,n)*1000.
 p.y=randomu(seed,n)*1000.
 p.z=randomu(seed,n)*1000.
 return,p
end

pro splitsearch2,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=3    ;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)

 if n gt threshold then begin
   hx=histogram(p.x,nbins=n_split+1,locations=x,reverse_indices =rx) ;1-D
binning
   hy=histogram(p.y,nbins=n_split+1,locations=y,reverse_indices =ry)
   hz=histogram(p.z,nbins=n_split+1,locations=z,reverse_indices =rz)

  for i=0,n_split-1 do begin
   for j=0,n_split-1 do begin
    for k=0,n_split-1 do begin
     qx=[rx[rx[i]:rx[i+1]-1]] ;indices from 1-D x bins
     qy=[ry[ry[j]:ry[j+1]-1]]
     qz=[rz[rz[k]:rz[k+1]-1]]
     q=where(histogram([qx,qy,qz],min=0) eq 3) ;indices from 3-D bins
(i.e. where all of x,y,z
                                    ;lie inside subvolume

     if n_elements(q) ge 2 then splitsearch2,p[q],dist ;splitsearch
again, if enough to compare
    endfor
   endfor
  endfor

 endif else begin
```

```
  q1=rebin(indgen(n),n,n) ;set up indices for pairwise comparison
  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 neighbours (q1>q2
to avoid reverse pairs and q1=q2)
  if i[0] ne -1 then print,p[q1[i]],p[q2[i]] ;print the neighbours, if
found
 endelse
end

IDL> p=ss_MakeGalaxies(1e6)
IDL> splitsearch2,p,.1
{      98.4014      461.355      673.278}{      98.4379      461.400
  673.197}
{      190.002      444.629      924.567}{      189.966      444.613
  924.605}
{      254.499      823.541      494.128}{      254.492      823.492
  494.062}
{      638.107      100.159      240.530}{      638.083      100.108
  240.606}
{      720.787      820.032      405.215}{      720.754      820.070
  405.199}

Greg
```

Subject: Re: fast search
Posted by greg michael on Wed, 18 Oct 2006 15:25:04 GMT
View Forum Message <> Reply to Message

All but one indestructible loop removed. Wish I had a use for this program now... hope it's useful to you, Marie!

```
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)

 if n gt threshold then begin
  mxx=max(p.x,min=mnx)
  mxy=max(p.y,min=mny)
```

```
  mxz=max(p.z,min=mnz)
  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)
  b=bx+by*n_split+bz*n_split^2
  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
    q=[ri[ri[i]:ri[i+1]-1]]
    if n_elements(q) ge 2 then splitsearch3,p[q],dist ;splitsearch
again, if enough to compare
   endif
  endfor

 endif else begin
  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 reverse pairs)
  if i[0] ne -1 then print,p[q1[i]],p[q2[i]] ;print the neighbours, if
found
 endelse
end

IDL> p=ss_makegalaxies(5e6)
IDL> splitsearch3,p,.02
{      915.876      843.515      319.991}{      915.861      843.528
  319.994}
```

Subject: Re: fast search
Posted by m.goullant@gmail.com on Thu, 19 Oct 2006 10:46:07 GMT
View Forum Message <> Reply to Message

Hi Greg,
thank you very much, for your time!

I even dont understood the second algorithm and you already have the
3!! :-(. In this weekend i'll take a better look in your explendid
work!

I'm a newbie in IDL, so I have to understand some functions that you
use! I
If is not ask to much, can you coment all the lines of your code?

What I want is a point to points search, so I am trying to understand
in your "quadtree" technique where I can adapt to my needs. I only need
to compare (x,y) - 2D and get the Z values to compare

This is more and less What I want to do:

 have this geographic data (could be 100000, 2 million, 4 million
depends):

```
PRO example


points = myData()

;data structure of an irregular point cloud
x = points.x ;X coord
y = points.y ;Y coord
z = points.z ; Elevation


n = 5 ; number of iterations
mask = 4.0 ; diameter. in meters. Like your dist
maskType = 0 ; 0 a circle, 1 square

FOR i=0L,n-1 DO BEGIN
        newZ = erosion(x,y,z,dist,MaskType)
        ;(...)
        dist = dist + 2
ENDFOR

END

FUNCTION erosion,x,y,z,mask,maskType ; Apply erosion to the data

  newZ=z
  radio = mask /2
  FOR i=0L,N_ELEMENTS(z)-1 DO BEGIN
                kernel = applyKernel(x,y,z,i,radio,maskType)
;center the kernel in the data(i) and get neighbours there are inside
of the mask
                newZ[i]=MAX(z[kernel])
  ENDFOR
  RETURN,newZ

END
```

```
FUNCTION applyKernel,x,y,z,i,dist,maskType

  square = WHERE(x LE x[i] + dist AND x GE x[i] - dist AND y LE y[i]
+ dist AND y GE y[i] - dist)

  IF (maskType EQ 0) THEN BEGIN
        sqDistance = sqrt((x[square] - x[i])^2 + (y[square] -
y[i])^2)
        neighbors = WHERE(sqDistance LE dist)
        circle = square[neighbors]
        RETURN,circle
  ENDIF

  RETURN,square
END
```

Because this is an iterative process with a "dist" variable, it's posible implement your code?

Thank's in advance,
Marie

Subject: Re: fast search
Posted by m.goullant@gmail.com on Thu, 19 Oct 2006 11:01:18 GMT
View Forum Message <> Reply to Message

Hi Paolo,

thanks for the article, i'll read cautiously!

regards,
Marie

Paolo Grigis wrote:
> This link may be useful to you:
>
> http://www.dfanning.com/code_tips/pts_in_sphere.html
>
> Ciao,
> Paolo
>
>
> m.goullant@gmail.com wrote:
>> Hi there,
>>
>> I have the following problem:

>>
>> ;data structure of an irregular point cloud
>> x = points.x
>> y = points.y
>> z = points.z
>>
>> search radio
>> radio = 8
>>
>> FOR i=0L,N_ELEMENTS(z)-1 DO BEGIN
>>
>> square = WHERE(x LE x[i] + radio AND x GE x[i] - radio AND y LE
>> y[i] + radio AND y GE y[i] - radio)
>> ;(...)
>>
>> ENDFOR
>>
>> I realize that WHERE will do the job, but at very low efficiency.
>> WHERE
>> makes no assumptions about the list being ordered. It seems to me it
>> has
>> to check every element of the array, requiring N steps for an N-element
>> array
>>
>> There is a faster way to do this?
>>
>> thanks,
>> Marie
>>
