Subject: Re: TRIANGULATE. Finding contiguous cells efficiently?
Posted by Libertan on Mon, 26 Feb 2007 03:37:54 GMT
View Forum Message <> Reply to Message

Wox, thanks for your time and effort. I wrote a lengthy reply which either didn't get posted or was sent to your email address. In case it has gone into the abyss, I did some timing experiments and your loop routine was 10% faster than my routine (similarish but less compactly written) for several thousand delaunay cells (run time ~170 seconds for 10^4 nodes). The ingenious vector routine you rote obviously can't handle that many cells. I should've mentioned this before.

thoughts:
1) The connectivity list might be fruitful after all.
2) Also I think that in these looped codes, the accumulated info in the loop can be used to increasingly reduce the remaining workload.

I'd like to keep this topic alive, so I'll post any significant progress. Anyone else feeling inspired?

Thanks again Wox

Subject: Re: TRIANGULATE. Finding contiguous cells efficiently?
Posted by Wox on Mon, 26 Feb 2007 12:27:49 GMT
View Forum Message <> Reply to Message

On 25 Feb 2007 19:37:54 -0800, "Libertan" <tbethell@umich.edu> wrote:

> In case it has gone into the abyss

It has :-)

> thoughts:
> 1) The connectivity list might be fruitful after all.

I don't think this could make things faster.

> 2) Also I think that in these looped codes, the accumulated info in
> the loop can be used to increasingly reduce the remaining workload.

Well ... one could think of things like: "deleting triangles which are already used 3 times". Example below gives an 18% improvement on my PC, when using the shrinking-triangles.

```
pro test
x=RANDOMU(seed,10000,/normal)*10
y=RANDOMU(seed,10000,/normal)*10
nnodes=n_elements(x)
TRIANGULATE, X, Y, triangles
s=size(triangles,/dimensions)

; 1. With loop
Tm=systime(1)
ctriangles=fltarr(s[0],s[1],3)
for i=0,s[1]-1 do begin
 t=total((triangles eq triangles[0,i])+$
 (triangles eq triangles[1,i])+$
 (triangles eq triangles[2,i]),1)

 ind=where(t eq 2,ct)
 if ct ne 0 then ctriangles[*,i,0:ct-1]=triangles[*,ind]
endfor
; Third dimension gives the three contiguous neighbours
print,systime(1)-Tm
Tm=systime(1)

; 2. With loop + shrinking triangles
Tm=systime(1)
ctriangles=fltarr(s[0],s[1],3)
trshrink=triangles
nused=bytarr(s[1])
indtr=lindgen(s[1])
for i=0,s[1]-1 do begin
 t=total((trshrink eq triangles[0,i])+$
 (trshrink eq triangles[1,i])+$
 (trshrink eq triangles[2,i]),1)

 ind=where(t eq 2,ct)
 if ct ne 0 then begin
  ctriangles[*,i,0:ct-1]=trshrink[*,ind]
  nused[ind]++

  tmp=where(nused eq
3,ct,COMPLEMENT=ind,NCOMPLEMENT=ct2)
  if ct ne 0 then begin
   if ct2 ne 0 then begin
    nused=nused[ind]
    trshrink=trshrink[*,ind]
   endif
  endif
 endif
endfor
```

```
; Third dimension gives the three contiguous neighbours
print,systime(1)-Tm
Tm=systime(1)
end
```

Subject: Re: TRIANGULATE. Finding contiguous cells efficiently?
Posted by Libertan on Tue, 27 Feb 2007 19:44:13 GMT
View Forum Message <> Reply to Message

Wox,

Okay, here's an code which surpasses my expectations; it seems to be
over 10,000% faster. I kid ye not. initially vectorized, ends with
simple loop over uncostly operations. I thought the invocation of
SORT would be slow, but imagine it's written in C. would I be right
in thinking that the overall trick is to use 1) as few operations as
possible, 2) use vectorized forms, and 3) use IDL instrinsics written
in C?

I'm sure it's not particularly well written, but what do you think?
Writing fast codes in IDL is a tricky business. Wox, if you care and
are prepared to email me your name, I'll acknowledge our discussion
when/if I publish future results (in Astrophysical Journal).

```
x=randomu(seed,np)
y=randomu(seed,np)
TRIANGULATE, X, Y, trang
s=size(trang,/dimensions)
ntr=s(1)
trang=trang+!pi  ;perhaps unnecessary, but ensures all values are
greater than 1. see below.

 ;============================================================ =====
;
;Crux: make array of edges, instead of pairs of vertices. Using the
two vertices of each edge
;create a *single* unique 'value' for the edge.  Use fast SORT to find
pairs of like edges.  Address
; simlarly sorted listr of cells to solve problem. Here I have
(foolishly) chosen to add the logs
; of the two vertices; Integer-> real.  yields unique value for the
edge?
; Instead, would something like x+1./y guarantee uniqueness?
;uniqueness of value ultimately limited by numerical precision?
 ;============================================================ =====
tred=dblarr(3,ntr)
tred(0,*)=double(alog(trang(0,*)))*double(alog(trang(1,*)))  ;need
arguments >1 ideally. see above
tred(1,*)=double(alog(trang(1,*)))*double(alog(trang(2,*)))  ; think
of better operation.
tred(2,*)=double(alog(trang(2,*)))*double(alog(trang(0,*)))

numtred=LONG(n_elements(tred))   ;=3*ntr

 ;============================================================ =====
; turn into vector of edges, instead of array, sort.
 ;============================================================ =====

edgvec=reform(tred,numtred)
celvec=LONG(findgen(LONG(3.*ntr))/3.)  ; =(0,0,0,1,1,1,2,2,2, etc.
Vector of cells associated
; with edgvec

edgsort=LONG(sort(edgvec, /L64))   ; sort order of edges
edgvecs=edgvec(edgsort)    ; sorted into edge value order
celvecs=celvec(edgsort)   ; likewise rearrange cells to keep track of
edge-cell relationship

;print, 'edgvec  :',edgvec
;print, 'celvec  :',celvec
;print, 'Now sorted...'
;print, 'edgvecs :',edgvecs
;print, 'celvecs :',celvecs

 ;============================================================ =====
; check that edges have unique values? skip.
 ;============================================================ =====

neigh=intarr(3,ntr)
neigh(*,*)=-1       ; any cells on convex hull will have one unpaired
edge -> -1

nedgvec=n_elements(edgvecs)
edgcount=LONG(intarr(ntr))

print, 'Starting loop over edges'

for i=0L, LONG(nedgvec-2) do begin   ; loop over edges, ordered by
their unique values.

        cellhere=LONG(celvecs(i))
        cellnext=LONG(celvecs(i+1))
        if(edgvecs(i) eq edgvecs(i+1)) then begin

                neigh(edgcount(cellhere),cellhere)=cellnext  ;this
cell

neigh(edgcount(cellnext),cellnext)=cellhere  ;complementary cell

                edgcount(cellhere)=edgcount(cellhere)+1L
                edgcount(cellnext)=edgcount(cellnext)+1L

        endif

endfor

end
```

Subject: Re: TRIANGULATE. Finding contiguous cells efficiently?
Posted by Libertan on Tue, 27 Feb 2007 20:27:13 GMT

View Forum Message <> Reply to Message

In the above, when I said "Instead, would something like x+1./y guarantee uniqueness?"
I meant A+1./B where A and B are vertex indices. The x,y coords of the vertices don't play a role at all.

Subject: Re: TRIANGULATE. Finding contiguous cells efficiently?
Posted by Wox on Wed, 28 Feb 2007 09:42:43 GMT

View Forum Message <> Reply to Message

Unique edges: nice thinking there :-). Off course, I would write some things differently like:

```
neigh=intarr(3,ntr)
neigh(*,*)=-1
=> neigh=replicate(-1,3,ntr)

edgcount(cellhere)=edgcount(cellhere)+1L
=> edgcount[cellhere]++
```

To make unique edges, you could use hash functions like CRC32 (with some adaption to make [a,b] the same as [b,a]), but assuming the vertices are stored in ulong's (i.e. 32bit), this will do:
c=(a>b)+ishft(long64(a<b),32)

Who am I: http://www.ua.ac.be/wout.denolf

On 27 Feb 2007 11:44:13 -0800, "Libertan" <tbethell@umich.edu> wrote:

> Wox,
>
> Okay, here's an code which surpasses my expectations; it seems to be
> over 10,000% faster. I kid ye not. initially vectorized, ends with
> simple loop over uncostly operations. I thought the invocation of
> SORT would be slow, but imagine it's written in C. would I be right
> in thinking that the overall trick is to use 1) as few operations as
> possible, 2) use vectorized forms, and 3) use IDL instrinsics written
> in C?
>
> I'm sure it's not particularly well written, but what do you think?
> Writing fast codes in IDL is a tricky business. Wox, if you care and
> are prepared to email me your name, I'll acknowledge our discussion
> when/if I publish future results (in Astrophysical Journal).

Subject: Re: TRIANGULATE. Finding contiguous cells efficiently?
Posted by Libertan on Thu, 01 Mar 2007 01:11:39 GMT

View Forum Message <> Reply to Message

Yes, compact notation is not my forte, and I thought CRC32 was a type
of camera battery. Unimpressive, I know.
I'll give c=(a>b)+ishft(long64(a<b),32) a try. Sounds like something
worth understanding. Thanks again for the feedback.

Libertan.

Subject: Re: TRIANGULATE. Finding contiguous cells efficiently?
Posted by Wox on Thu, 01 Mar 2007 14:39:58 GMT

View Forum Message <> Reply to Message

This just puts two 32bit numbers in one 64bit number. The lowest value
is stored in the lower-order dword and the highest in the HO DWORD.
Without the '<' and '>', edge [a,b] wouldn't be the same as edge
[b,a].

No I see there is a mistake :-). It must be
c=(a>b) or ishft(long64(a<b),32)

On 28 Feb 2007 17:11:39 -0800, "Libertan" <tbethell@umich.edu> wrote:

> I'll give c=(a>b)+ishft(long64(a<b),32) a try. Sounds like something
> worth understanding.

Subject: Re: TRIANGULATE. Finding contiguous cells efficiently?
Posted by Wox on Thu, 01 Mar 2007 14:44:55 GMT
View Forum Message <> Reply to Message

Then again, using '+' or 'or' gives the same result off course. I got confused there when explaining the HO|LO business.

On Thu, 01 Mar 2007 15:39:58 +0100, Wox <nomail@hotmail.com> wrote:

> No I see there is a mistake :-). It must be
> c=(a>b) or ishft(long64(a<b),32)
>
> On 28 Feb 2007 17:11:39 -0800, "Libertan" <tbethell@umich.edu> wrote:
>
>> I'll give c=(a>b)+ishft(long64(a<b),32) a try. Sounds like something
>> worth understanding.