

Re: Regularly populating the line – revised version

Source: <http://sci.tech-archive.net/Archive/sci.math.num-analysis/2005-01/0493.html>

From: Lord Snooty (*bonzo_at_dog.com*)

Date: 01/30/05

Date: Sun, 30 Jan 2005 20:56:19 GMT

Thanks (for a reply somewhat snootier than my handle suggests).
As I posted, I did hit the books before posting this, but you do raise good points despite your apparent reading impairment.
(Since you impugn my academic curiosity, you will have to deal with feedback like this).

I think you may also have a writing issue; $(i-i)$ is identically zero, so your inequality makes no sense.
Perhaps it is simply a typo.

It is also a tautology that i^*p/n changes. It changes for every value of i , irrespective whether the 'if' fires or not.
So I have no idea what you meant by that. Another typo?

However, your observation – which is correct – that $a = i^*p \bmod n$ may allow me to work up a proof.
This is still necessary because you haven't shown that the 'if' fires exactly p times, which in fact is the question I posed.

It is indeed the case that the algorithm is very reminiscent of Bresenham (in a former life I designed 3D graphics chips).
However, this algorithm comes from a rather different domain having no connection with scan lines.
The way I introduced it stands on its own; it's a populating algorithm, I suppose one could call it, of a completely general nature.

LS

"Gordon Sande" <g.sande@worldnet.att.net> wrote in message
news:PR6Ld.78622\$Qb.34951@edtnps89...

>
>

> *Lord Snooty wrote:*

>> *Due to a typo, one more time with feeling, and less C-stylistic too:*

>>

>> *for (i=a=0; i < n; i++) {*

>> *a = a + p;*

>> *if (a >= n) {*

```
>> //fill at position i
>> a = a - n;
>> }
>> // else leave position i unfilled
>> }
>>
>> It's trivial to analyse this for  $(n \bmod p) = 0$  of course.
>> I'm only interested in nonzero  $(n \bmod p)$ .
>>
>
>
> The first observation is that a is just  $i*p \bmod n$ .
> Then one wants to know what make the if condition true.
> It is true whenever the value of  $i*p/n$  (integer divide)
> changes. Otherwise stated as  $(i-i)*p/n \neq i*p/n$ . This
> starts to look like the algorithm for drawing a diagonal
> line on an integer lattice, which is Bresenham's algorithm
> if you are doing computer graphics. A full citation
> is left as an exercise for the reader. Some of the dates will
> be in the prehistoric period BG (Before Google).
>
> The curious handle does not suggest a strong academic interest
> so one might even guess that the original problem is computer
> graphics. The ancients did have considerable wisdom and the even
> more curious habit of recording much if it in books. So the
> obvious thing is for Lord Snooty to try reading some books.
>
>
>>
>> "Lord Snooty" <bonzo@dog.com> wrote in message news:...
>>
>>> It's not too uncommon to want to populate n bins with a given number p of
>>> objects, such that they are as regularly spaced as possible – we seek to
>>> avoid
>>> bunching, in other words. For example, if  $n=8$  and  $p=2$ , then trivially we'd
>>> seek one of the cyclic permutations of  $PxxxPxxx$ , where P represents one of
>>> the
>>> p objects and x represents an unfilled bin.
>>>
>>> I came across the following very efficient algorithm recently, which works
>>> on
>>> all tested integer pairs  $(n,p)$ ,  $n \geq p$ . Trouble is, I have difficulty
>>> proving
>>> that it will always emit exactly p objects. I'd appreciate any attempts at
>>> proving this. I'd also like to know if it has a name, because I've never
>>> come
>>> across it before. I can't find it in Knuth, for example, because I can't
>>> find
>>> a good name for it with which to do an index search. Here's the algorithm
>>> (C-code style):
>>>
```

```
>>>for (i=a=0; i < n; i++) {  
>>> a += p;  
>>> if (a >= n) {  
>>> //fill at position i  
>>> a -= nN;  
>>> }  
>>> // else leave bin unfilled  
>>>}  
>>>  
>>>  
>>  
>>
```