Marmakoide's Blog

My coding journeys, let me show you them

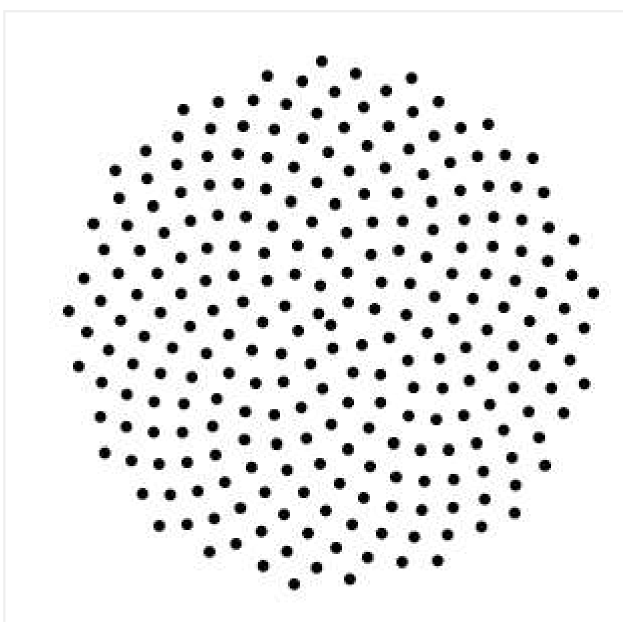
Spreading points on a disc and on a sphere

Posted on **April 4, 2012**

When working with computer graphics, machine vision or simulations of various kind, two problems (among many others) which keep popping out are

- How to have N points approximately evenly distributed over a disc
- How to have N points approximately evenly distributed over a sphere

One way to build a solution for those two problems is to rely on a particle system. Each point is a particle, particles are repulsing each other with a $\frac{1}{r^2}$ force. Put the particles on the disc or the sphere surface, at random locations. Shake vigorously the particles, add some small drag force, let the particles reach a steady state, shake again ... rinse, repeat. It is not as easy as it sounds. How to setup the forces ? How to handle the boundary of the disc ? Which integrator to use to compute the particle's motion ? And for more than a few hundred particles, it will be terribly slow.



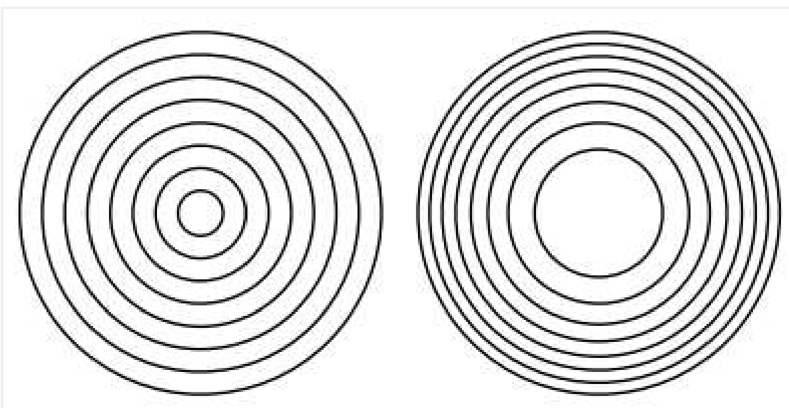
— 256 points with Vogel's method

There are much simpler and leaner algorithms to evenly distribute N points over a disc or a sphere, if one can live with just a good approximation. The main idea fits in one word : *spiral* ! Let's start with a disc. Imagine a spiral of dots starting from the center of the disc. In polar coordinates, the N points are produced by the sequence

$$\rho_i = \theta i$$

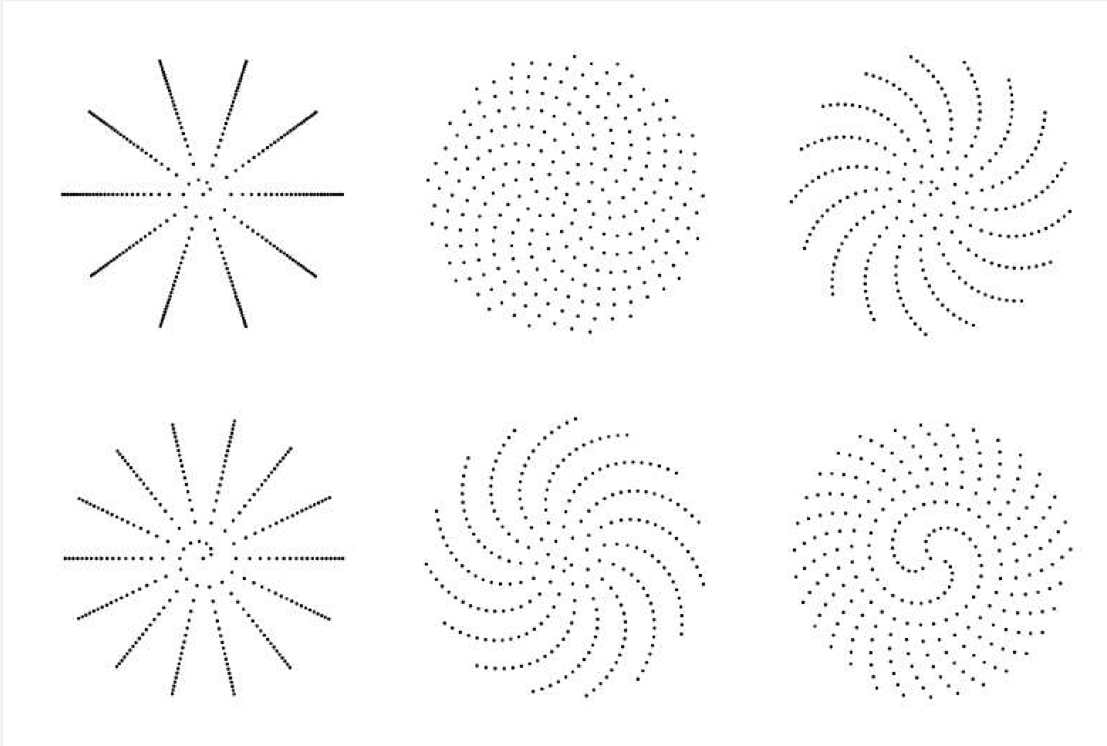
$$\tau_i = \sqrt{\frac{i}{N}}$$

ρ_i and τ_i are respectively the angle in radian and the radius of the i-th point. Why $\tau_i = \sqrt{\frac{i}{N}}$? Let's try to cut a unit disc in one disc and one ring of equal areas, by cutting the unit disc at radius r_c. The small disc and the ring are of equal areas, thus $\pi r_1^2 = \pi - \pi r_1^2$. Thus $r_1 = \sqrt{\frac{1}{2}}$. Now, let's try to cut the unit disc in one disc and 2 concentric rings, all of equal areas. A slightly more complex calculation (a system of 2 linear equations) would tell us that we should cut the unit disc at radius $r_1 = \sqrt{\frac{1}{3}}$ and $r_2 = \sqrt{\frac{2}{3}}$. The calculation is a bit more complex when cutting the unit disc in N equal areas rings, but you can guess the answer now : $r_1 = \sqrt{\frac{1}{N}}$, $r_2 = \sqrt{\frac{2}{N}}$, ..., $r_i = \sqrt{\frac{i}{N}}$.



— Equal thickness versus equal areas concentric rings

What about the ideal θ ? Brace yourself... This is the golden angle, $\pi(3 - \sqrt{5})$! It's roughly equal to 137.508°, or about 2.39996 radians. The golden angle is the *"most irrational"* angle, defined as $2\pi(1 - \frac{1}{\phi})$ with ϕ being the golden ratio. If θ is a rational number, we would obtain clusters of points aligned with the center of the disk. Thus θ have to be irrational. As it turns out, the golden ratio is the irrational number the hardest approximate with a continued fraction. Written as a continued fraction, the golden ratio is the irrational number with the slowest convergence of all the irrational numbers. The golden angle gives the best possible spread for the ρ_i angles. This method to spread points over a disc is called *Vogel's method*.



— Effect of the angle step parameter, on 256 points. The 2 leftmost points layout are for rational values of the angle step. The top center one is for square root of 2, the 3 others are for others irrational values.

Vogel's method is dead simple to implement. In pure, barebone Python, it can be done like this

```
import math

n = 256

golden_angle = math.pi * (3 - math.sqrt(5))

points = []
for i in xrange(n):
  theta = i * golden_angle
  r = math.sqrt(i) / math.sqrt(n)
  points.append((r * math.cos(theta), r * math.sin(theta)))
```

Using numpy, one can use vector operations like this

```
import numpy

n = 256

radius = numpy.sqrt(numpy.arange(n) / float(n))

golden_angle = numpy.pi * (3 - numpy.sqrt(5))
theta = golden_angle * numpy.arange(n)

points = numpy.zeros((n, 2))
points[:,0] = numpy.cos(theta)
points[:,1] = numpy.sin(theta)
points *= radius.reshape((n, 1))
```

What about the sphere ? We can reuse the golden angle spiral trick. In cylindrical coordinates, we would generate N points like this

$$\rho_i = \theta i$$

$$\tau_i = \sqrt{1 - z_i^2}$$

$$z_i = \left(1 - \frac{1}{N}\right)\left(1 - \frac{2i}{N-1}\right)$$

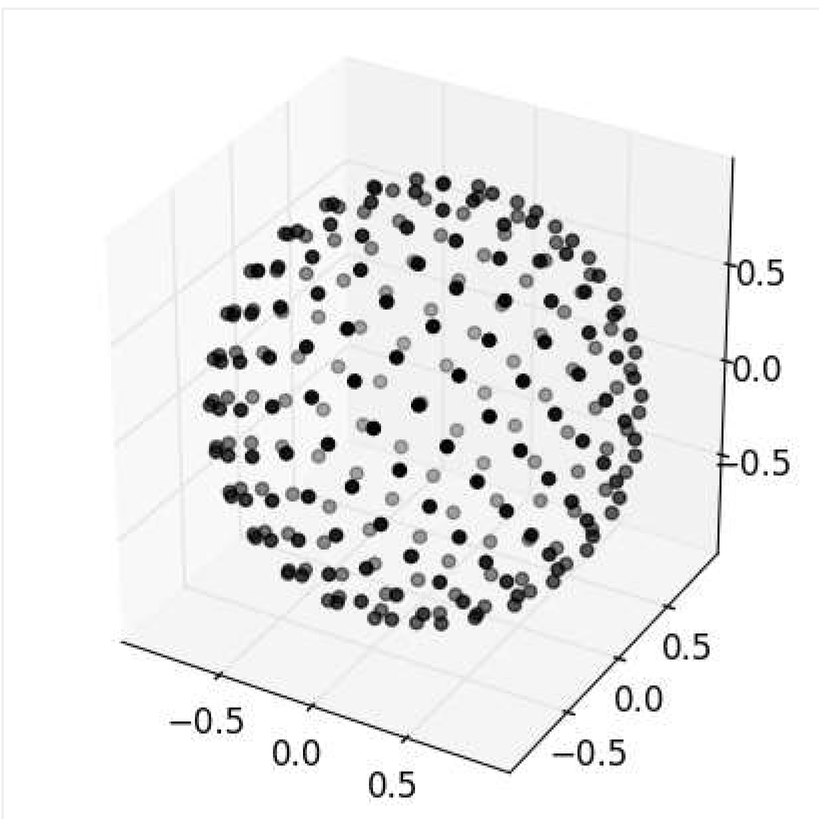
And *voila*, an extremely cheap method to spread point evenly over a sphere's surface ! The code for this method is very close to the one for *Vogel's method*.

```
n = 256

golden_angle = numpy.pi * (3 - numpy.sqrt(5))
theta = golden_angle * numpy.arange(n)
z = numpy.linspace(1 - 1.0 / n, 1.0 / n - 1, n)
radius = numpy.sqrt(1 - z * z)

points = numpy.zeros((n, 3))
points[:,0] = radius * numpy.cos(theta)
points[:,1] = radius * numpy.sin(theta)
points[:,2] = z
```

Note that you can also use this method to make a sphere mesh. The sphere will look good even with few polygons, but the indexing and computing the normals won't be especially fast. For this specific problem, making a sphere mesh, Iñigo Quilez have a very nice, efficient method introduced here



— 256 points on a sphere

This entry was posted in **computational geometry** by **marmakoide**. Bookmark the **permalink [http://blog.marmakoide.org/?p=1]** .

8 THOUGHTS ON "SPREADING POINTS ON A DISC AND ON A SPHERE"

Mr Speaker
on **May 2, 2012 at 9:00 pm** said:

Excellent article, but damn you - now I have to go and experiment with this and I'm not going to get ANYTHING done today!

> 
>
> **marmakoide**
> on **May 2, 2012 at 9:33 pm** said:
>
> Ha ha, I gonna troll you with more posts of that kind, then ^^

Pingback: Illuminated.js – 2D lights and shadows rendering engine for HTML5 applications • @GreWeb

greweb
on **May 12, 2012 at 1:01 am** said:

Thanks a lot man!
Your research helps me to figure out how to spread light point samples for a shadow casting engine I made: http://blog.greweb.fr/2012/05/illuminated-js-2d-lights-and-shadows-rendering-engine-for-html5-applications/

Regards

> 
>
> **marmakoide**
> on **May 12, 2012 at 8:23 am** said:
>
> Glad it helped ! Very nice work you did there : elegant, sounds cheap to compute and it looks gorgeous... Indeed, sampling light sources is one of the usage I got in mind for this trick.

Travis
on **July 20, 2012 at 8:54 am** said:

This is very close to exactly what I was looking for, but to go a step further could you elaborate on how to describe the 3-axis coordinates [x,y,z] of any point among those evenly distributed on a sphere (say totaling 256 as in your example), without approximation? I am thinking since 256=4^4 we could look at this as (4^4)^n where n=1 and n can only be a whole integer, so given the understanding these points would spread symmetrically in every direction from the origin as n increases, is there a way to say this nicely when you have 256^(10^3) for instance?

> Travis
> on **July 20, 2012 at 8:56 am** said:
>
> More specifically, can you design the grid in increments of 256^n so the approximation is generated just by input of pi and/or the power of 256^n?

> **marmakoide**
> on **July 21, 2012 at 8:06 am** said:
>
> Without approximation, the exact best layout for N points ? Not as a simple, explicit formula... There might be such a formula, but honestly I would be very surprised if there's one. The only way I know to get the exact layout would be some expensive iterations, where points repel each other with some electric force. I've code for this... However, for some special N, yes, there might a nice formula. But I don't know it ^^ You might take a look at this, which seems somehow in the spirit of your idea.

Comments are closed.