The Sisyphus table is a programmable device that moves a metal ball magnetically through a bed of sand to produce patterns. This document attempts to describe the logic of the table and give hints on how to produce images that can be displayed on it.

**Physical Characteristics**

The table can be envisioned as a circle with radius of 1, the center being 0 and the outer edges 1. The table is designed to operate in polar format – so all positions that the ball is moved toward must be expressed in polar format - theta, rho. 0,0 is the center of the table and 0,1 π/2,1 π,1 and 2 π/3,1 would be the N,W,S,E points. (Note here than in “conventional” polar format that 0,1 would be the E side of the table) . See Illustration 1.

**File Structure**

The table is fed information to produce the patterns through a “.thr” file which consists of a text file of a sequential list of number pairs (in polar format - theta, rho) separated by a space.

All Files must start with a 0,0 or 0,1 position. The end pair of numbers in the file must end with rho value opposite of the rho value in the first pair (if you start with a 0,0 you end with a theta, 0)

The ball is positioned by the table “reading” the data set pairs and it moving the ball to the next point specified in the instruction file. There are 2 “types” of instructions that the table can understand based on the difference between the theta values in the next set of instructions in the data set.

1. “Erase spiral” A spiral pattern can be generated (always centered from the 0,0 point but can start at any rho value) by having 2 data pairs with a large theta difference. The easiest way to do this is a multiple of 2 PI (a complete revolution) as the difference in the second data point.

0 0

314.159 1

Produces the “Erase” file on the default track list (spiral from the center to the outside)

0 1

314.159 0

Would produce a spiral from the outside to the center

0 0

314.159 .5

157.0795 1

Would produce a thin spiral from the center halfway out and a thicker spiral to the edge

1. “Normal Movement” Data pairs with a short distance between which makes the ball move in a line or curve. The shorter the difference between points – the more precise the curve. When the difference in theta value is large between data pairs (see the illustration 2) the table will “curve the line to connect” - This is why large “jumps” need to be avoided in patterns.

**Table Notes**

The table sees its 0 angle as the top axis (not the right axis as in conventional polar drawings) So in effect there is a Pi/2 (90 degree) rotation performed on the image. Most people probably just orient their table to “fix” this – but if you are producing an image that has a top or bottom - it “turns” it when you execute the file. (I just rotate the image -90 to compensate).

The table also X- axis mirrors the image you input. If you have a symmetric pattern (as most tracks seem to be) you never notice this artifact. I wrote a text generator program and my initial output was a mirror version of the letters. You can “fix” this by just mirroring your image in the X axis as the .thr file.

The table is exquisitely sensitive in movement near the 0,0 (center) of the table and the ball will slow or wobble in its path. The only “fix” for this is to avoid that area like the plague OR to make the movement increments very tiny in the area around a radius of .2 from the center. It’s an artifact of the table that you have to compensate in your program for or you will get wavy lines near there.

The 0 axis “ crossing flip” is an issue because it can produce a discontinuity between following theta values - the table making the ball take a circular path between points (instead of the straight path). The zero axis is where the angle can “jump” by a value of +- 2\*PI (depending on movement direction) and you have to be certain that you do not “jump” in the theta angles between points in your output file.

In polar coordinates there can be multiple ways to specify a point unlike Cartesian (x,y) where there is only one pair for each distinct point. 0,1 is the same as 2\*Pi, 1 is the same as -2\*PI, 1. If you go from say 6.1,.5 to 6.3, .5 (from quadrant IV to quadrant I) the table would be happy. But if you went from 6.1, .5 to 0.-1682 (which is exactly the same point) the table will “jump” the ball in a circular path.

The table will not accept any data where rho value is greater than 1. The ball will hang on the edge and the table will protest – loudly. ( In Cartesian terms , the Square Root of (X\*X + Y\*Y) must always be less than or equal to 1)

Programming Tips

I have made images using only polar code but I find it much easier to produce an X Y version and just convert it to polar. (if you are making symmetric patterns, polar can be easier, but if you are making pictures or images, my brain doesn’t “see” polar as clearly as Cartesian) AND Cartesian is non-ambiguous in terms of location of the coordinates. (you are not having identical points at 2π or -2π or having to deal with + / - angles. )

If you take that approach – you will have to convert the image back into polar for the table to understand. Here is what I found works.

1. Write out your x,y image as pairs of points - similar to the .thr file format
2. Mirror the X axis points if your image is not symmetric. – this means in plain language, all of the X points get a sign reversal. If it was a .5, it becomes a -.5 and so forth.
3. Rotate the image –π/2 to compensate for the table going to rotate what you input by π/2. X(rot) = X\*cos(rotation) –Y\*sin(rotation)

Y(rot) = X\*sin(rotation) + Y\*cos(rotation)

1. Check to be certain that you do not exceed a rho of 1 on your X,Y points. Square Root(X\*X +Y\*Y) <= 1
2. Convert your X,Y to polar Rho = Squareroot(X\*X +Y\*Y) <= 1 Theta = arcCos(x/rho)
3. Now the tricky part – because the value of theta can be negative if it is in different quadrants – So I just work out what quadrant the x,y points were in and then “normalize” the theta value as a positive angle (between 0 – 2π) for each point. But now you have a situation where a point in quadrant 4 could be a value of almost 6.2 adjacent to a value in quadrant 1 that was near 0. The table will “jump” if it sees that in the thr file.
4. So I go back and walk through the points and look where the cross of the 0 axis occurs (the 2π – 0 boundary) if it is a positive direction (quad 4 to 0) – I add 2π . if it is negative )quad 1 to 4), I subtract 2π. (and the value is additive so if I spin around again – it adds or subtracts the multiple to the next theta.

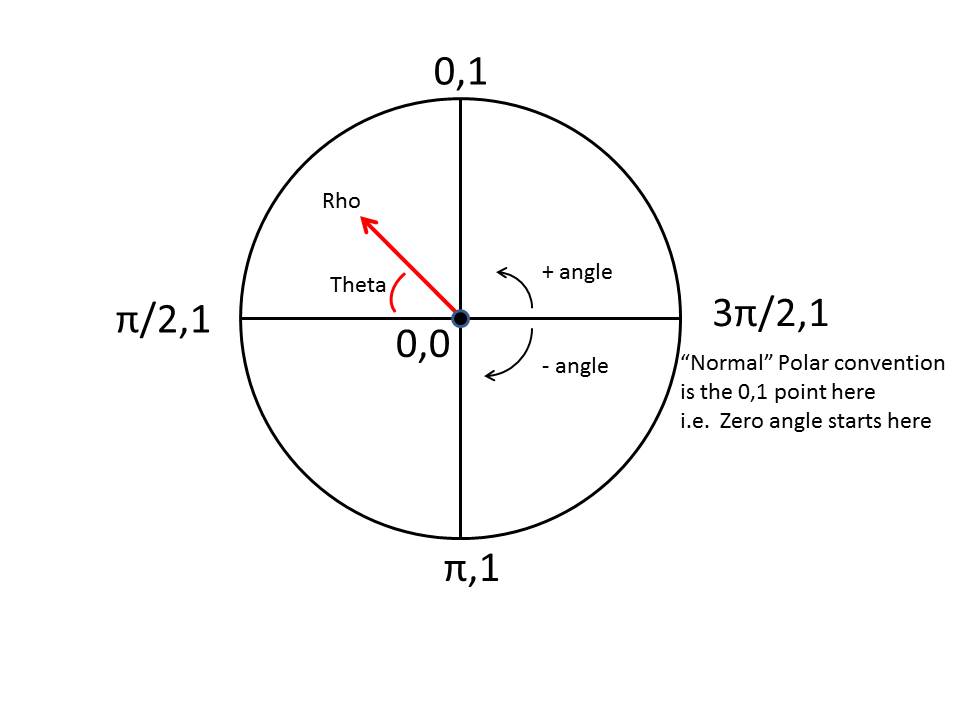
There are different ways to do this compensation, but this method seems to always work. Normalizing the angles AND then doing the jump up or down depending on how you cross the 0 axis. I get smooth motion and no angle jumps and my theta values are smooth positive or negative runs.

8 ) To finish, I generally add an “erase” pattern as the first 2 lines in the output file (to clear the table) and then go back and add whatever theta value line 2 was to ALL of the theta values in the set. I then write the values of the image to the file as line 3 to XX. The last line of the code just moves the ball to the center 0,0 or the edge theta,1 depending on where your image ends.

Ray Solomon

9-7-2019.

Illustration 1 - Table Geometry Illustration



**Illustratio**n **2 & 3 Showing Spacing effects between points and table output**

Notice how at wider point spacing the table will wobble then “jump & curve” between points.

