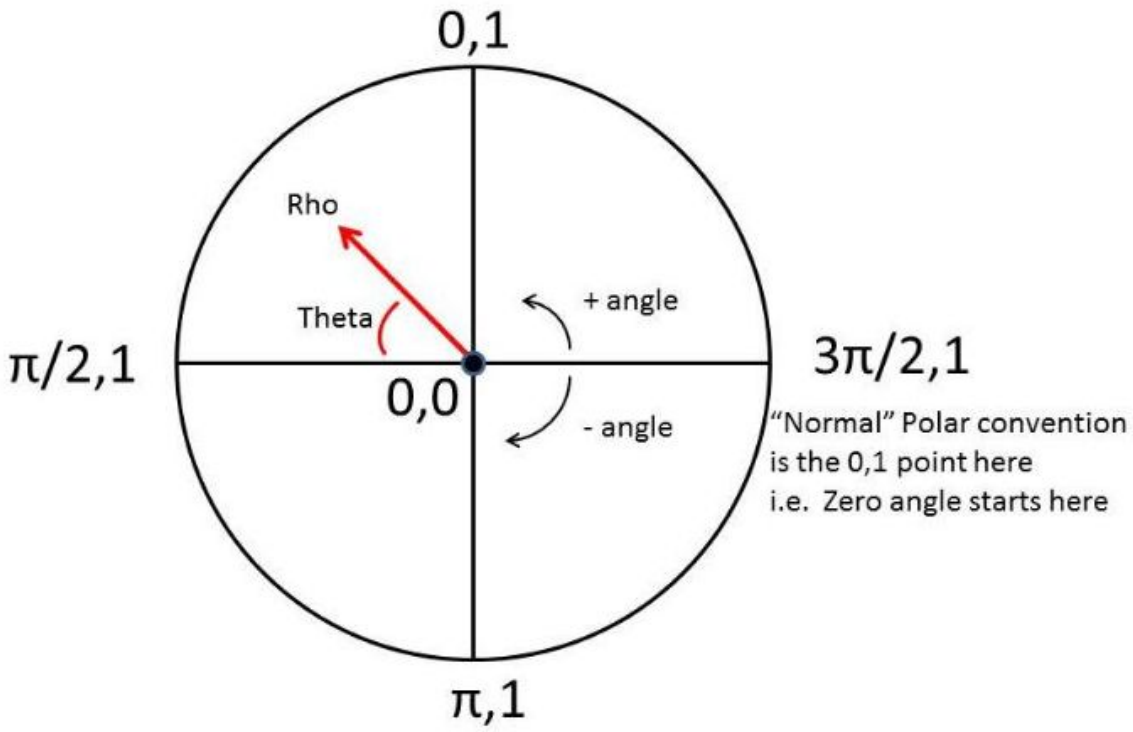
The Sisyphus table is a programmable device that moves a metal ball magnetically through a bed of sand to produce patterns. This document attempts to describe the logic of the table and give hints on how to program output that can be displayed on it.

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 instructions for moving the ball must be expressed in polar format as a series of theta, rho values. 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 below. All Sisyphus tables, regardless of size, use the same convention, so that instruction files can be played on any of them (ball size changes with table size).



In practical terms for programming the convention of having the 0 axis at the top means any image you produce in "conventional" layout – 0 axis on the right – is rotated π/2 (90 degrees) when it executes.

File Structure

The table is programmed to produce the patterns through a ".thr" file which is just a plain text file consisting of a sequential list of number pairs (in polar format - theta, rho) separated by a space. Comments can be added to the program listing by starting the line with a # character.

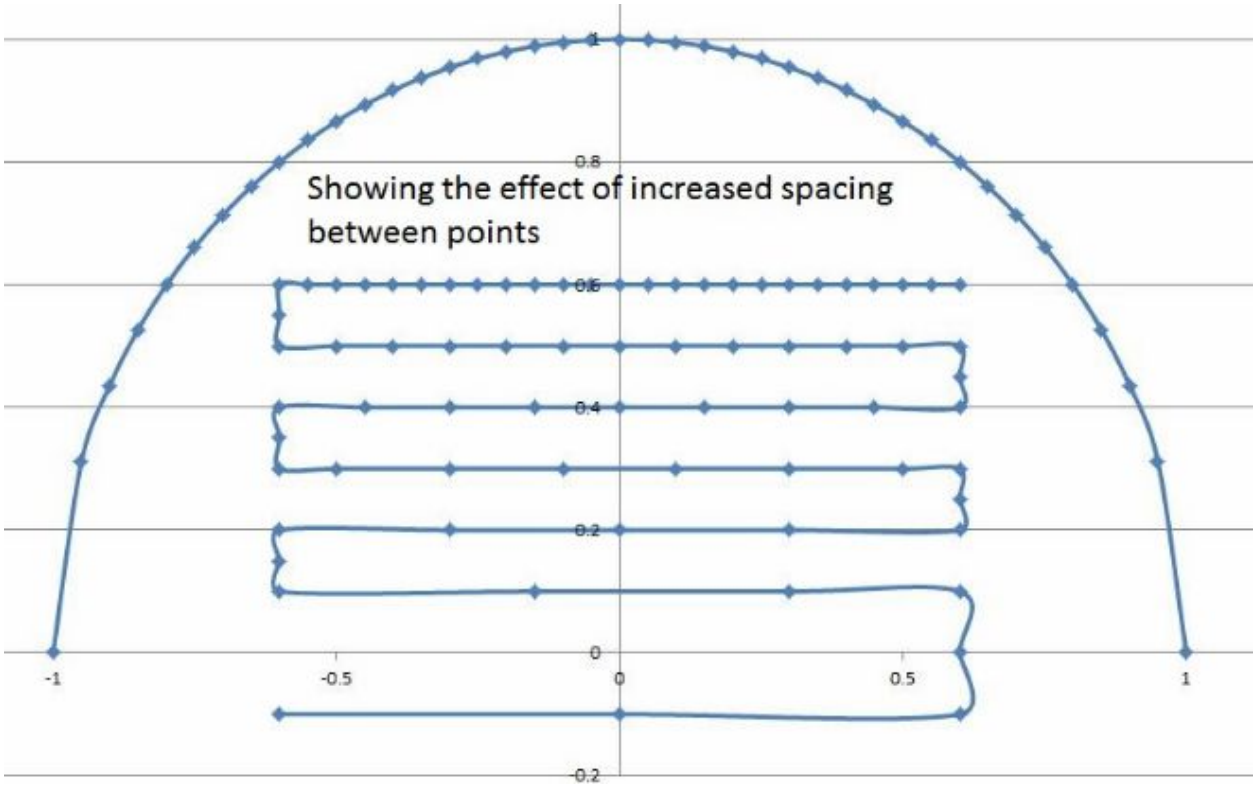
All Files must start and end with a rho value equal to 0 or 1.

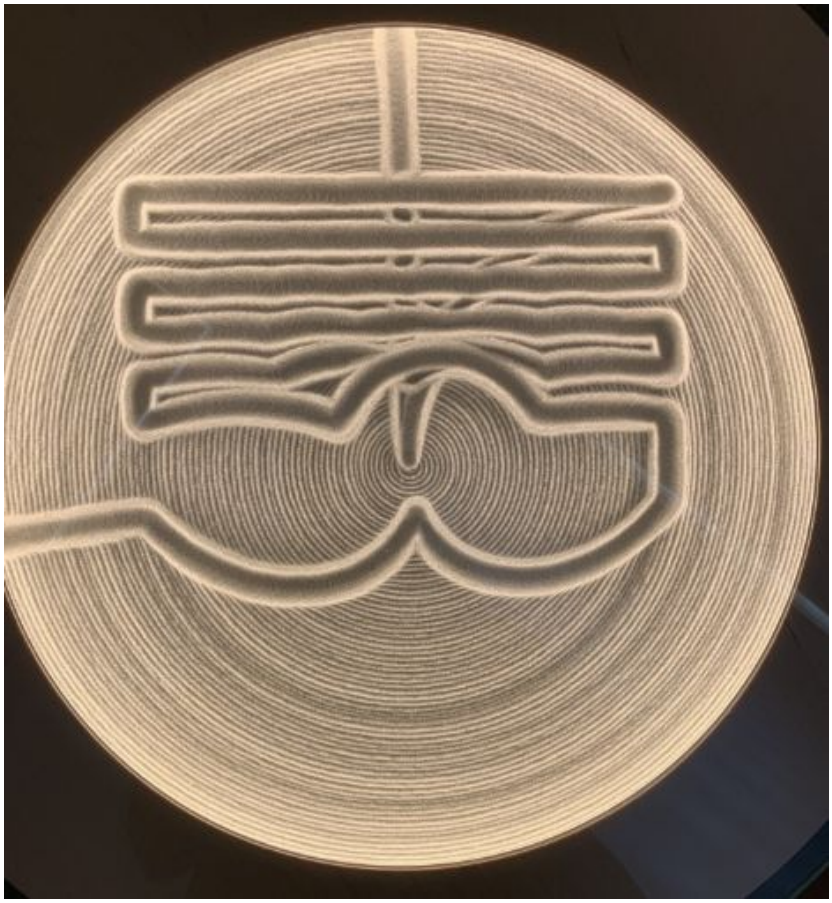
Theta value for the start or end point does not matter – BUT the end theta is relative to the value of the initial theta. (in simple terms – the theta values in the set must be a smooth flow (small differentials) or else you will have "jump spirals" in your paths.

The ball is positioned by the table "reading" the data set pairs and then moving the ball to the next point specified in the instruction file. The ball is always moved in a spiral path from one point to the next but this manifests as 2 "types" of instructions in practice. (Apologies to Bruce, but the wide difference of the path behavior makes describing the apparently different behavior necessary)

1) "Normal Movement" Data pairs with a short distance between them is used to make 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.

The illustrations show the path (as x,y) and the corresponding table output. Notice how at wider point spacing the table will wobble then "jump & curve" between points. (table output was rotated by $-\pi/2$ to produce the same orientation as the X,Y graph)





2) "Erase spiral" A spiral pattern can be generated by having 2 sequential data pairs with a large theta difference. The spiral is always centered from the 0,0 point –regardless of where it starts on the table - i.e. – you can't draw a spiral with its center not centered on 0,0 using this technique. The spiral may start at any rho value. The easiest way to do this is to use some multiple of 2 π (a complete revolution) as the difference in the second data point.

   The "Erase" file on the default track list (spiral from the center to the outside) is coded as:

   0 0
   314.159 1

   (make 50 rotations between rho 0 and 1)

   To produce a spiral from the outside to the center:

   0 1
   314.159 0

   To illustrate that the spiral effect can be repeated:

0	0	Start
31.41593	.5	Section 1
157.0796	.7	Section 2
628.3185	1	Section 3
Earlier erase		Section 4

(I stopped the ball at rho >.8 as the fine erase was very long)

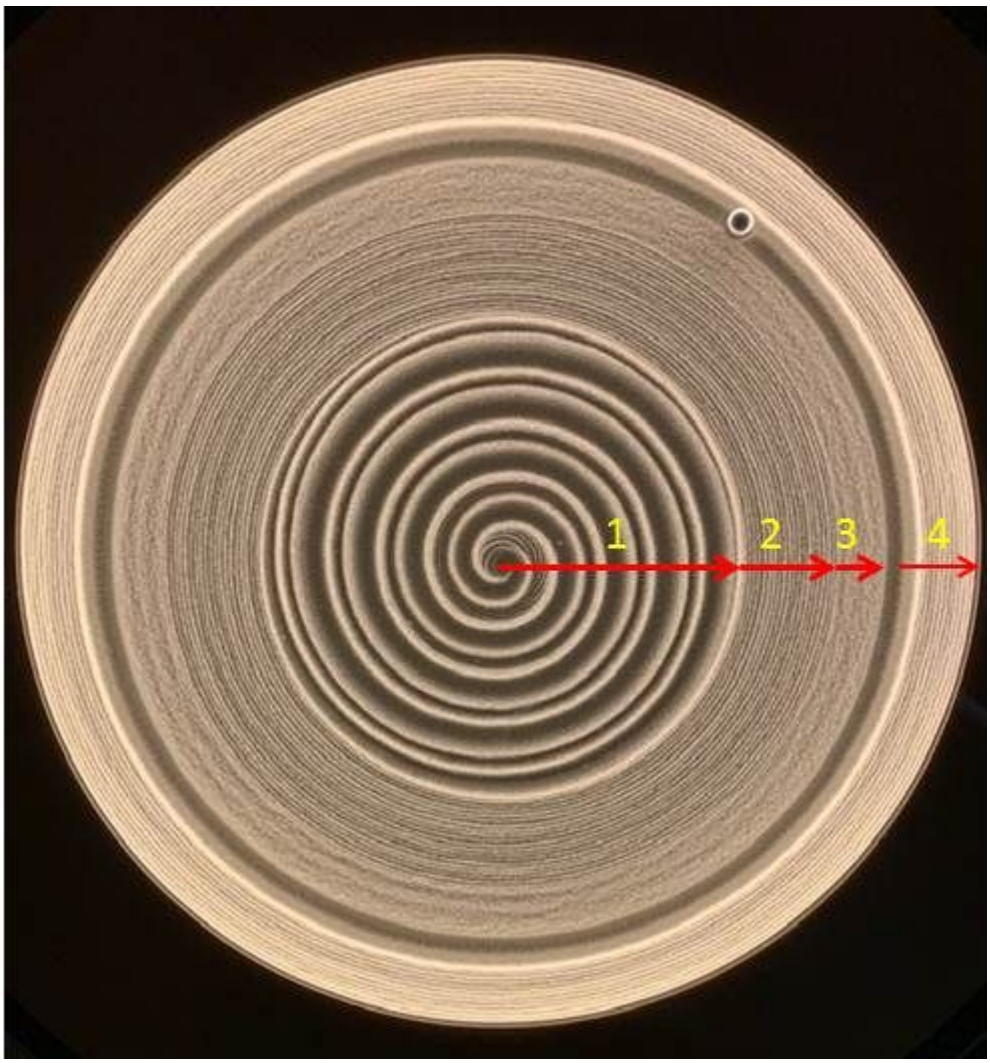


Table Notes

1) The table sees its 0 angle as the top axis (not the right axis as in conventional polar notation) So in effect there is a π/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 – the table “turns” it when you execute the file. (I just rotate the image -90 degrees to compensate).

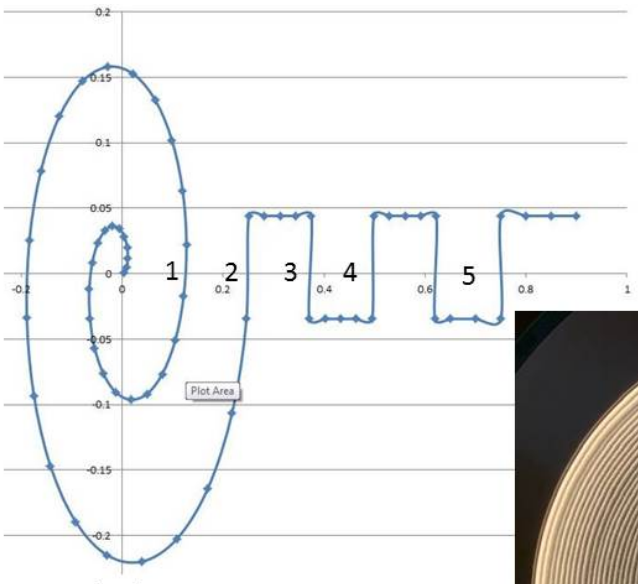
2) The table also X- axis mirrors the image you input. If you have a symmetric pattern (as most tracks seem to be) you never notice this artifact. I wrote a text generator program and my initial output was a mirror version of the letters. You can “fix” this by just mirroring your image in the X axis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

3) 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 rho value 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.

4) 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)

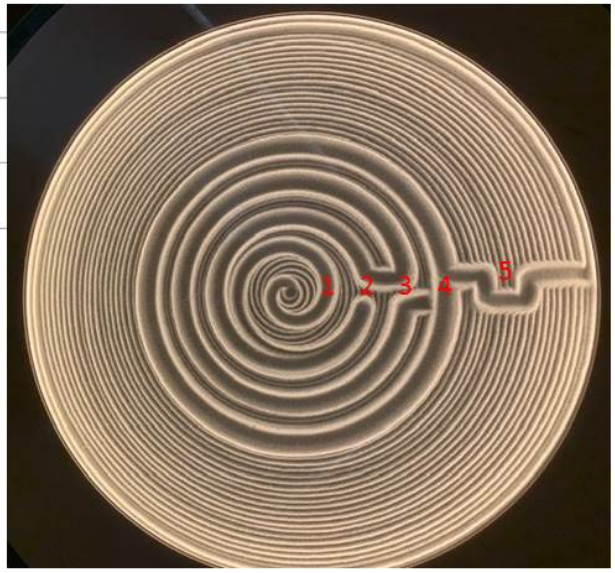
5) The 0 axis “crossing flip” is a huge issue because it can produce a discontinuity between points - the table making the ball take a spiral path between points (instead of the desired path).

> Why does this happen? Because if you are rotating a (positive) path around the table and reach a value of 2 π, if you continue in a positive direction your theta is now near 2 π on one side (quad 4) and near 0 on the other (quad 1). Reverse that if you are in negative angles. You must keep count of the movement across the 0 axis and add or subtract 2π to your angle depending on direction of cross to keep the theta differential smooth.
>
> 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*π, 1 is the same as -2*π, 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.
>
> You can see in the illustration below what happens when the theta differential is too large – the ball spirals around the table to connect the points and leaves a gap.



XY Path of Crossing 0 Axis

Table Output of 0 Axis Cross



This table is the values in the .thr file – the "good" 0 axis cross points are shown in yellow. "Bad" in red

	theta	rho			theta	rho			theta	rho	
1	0.17453	0.00349		22	6.77188	0.13544		42	6.43889	0.28534	
2	0.48869	0.00977		23	7.08604	0.14172		43	6.4237	0.31594	
3	0.80285	0.01606		24	7.4002	0.148		44	6.4112	0.3466	
4	1.11701	0.02234		25	7.71436	0.15429		45	6.40073	0.37731	
5	1.43117	0.02862		26	8.02851	0.16057		46	12.4733	0.37217	X3
6	1.74533	0.03491		27	8.34267	0.16685		47	12.48047	0.40315	
7	2.05949	0.04119		28	8.65683	0.17314		48	12.48662	0.43416	
8	2.37365	0.04747		29	8.97099	0.17942		49	12.49194	0.46518	
9	2.68781	0.05376		30	9.28515	0.1857		50	12.49661	0.4962	
10	3.00197	0.06004		31	9.59931	0.19199		51	-12.4778	0.50042	X4
11	3.31613	0.06632		32	9.91347	0.19827		52	-12.483	0.53124	
12	3.63028	0.07261		33	10.22763	0.20455		53	-12.4876	0.56208	
13	3.94444	0.07889		34	10.54179	0.21084		54	-12.4917	0.59293	
14	4.2586	0.08517		35	10.85595	0.21712		55	-12.4954	0.62378	

15	4.57276	0.09146		36	11.17011	0.2234		56	-12.6222	0.6204	X5
16	4.88692	0.09774		37	11.48427	0.22969		57	-12.6195	0.65092	
17	5.20108	0.10402		38	11.79843	0.23597		58	-12.6157	0.70085	
18	5.51524	0.1103		39	12.11259	0.24225		59	-12.6125	0.7508	
19	5.8294	0.11659		40	12.42674	0.24853		60	-12.5074	0.7513	X5b
20	6.14356	0.12287		41	6.45772	0.25482	X2	61	-12.5111	0.80122	
21	6.45772	0.12915	X1					62	-12.5144	0.85115	
								63	-12.5172	0.90109	
								64	-12.5198	0.95103	

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 X,Y 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)
4) 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
5) Convert your X,Y to polar Rho = Squareroot(X*X +Y*Y) <= 1 Theta = arcCos(x/rho) (See below for the discussion of correction of this to proper +/- value)
6) Now the tricky part – because the value of theta can be negative if it is in different quadrants – S 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. This means that I start with all theta values as positive and between 0 – 2π.

   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.
7) 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.

The other method to do this is to go back and construct differentials between each theta value and the one that follows and then add them up starting with the initial theta as the base. You still must deal with the crossing problem noted above when you do this. (and your angles better be consistently positive or negative to get the smallest correct differential between points).

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.

I want to add this discussion about converting from X,Y to polar because the Arc function does not always give the correct answer if you use it blindly.

Cartesian/ Polar conversion - the Arc Function Conversion trap.

As noted in 5) above Rho = Squareroot(X*X +Y*Y) (the value of which MUST be <= 1) and Theta = ArcCos(x/rho) OR Theta = ArcTan(y/x). BUT this will give the wrong value if your X,Y pair is in the 3rd or 4th quadrant. Look at the values below and you can see that just using the Arc function isn't enough if you want the correct theta value when you convert.

Quadrant	X	Y	rho	acos	atan
1	0.5	0.5	0.707107	0.785398	0.785398
2	-0.5	0.5	0.707107	2.356194	-0.7854
3	-0.5	-0.5	0.707107	2.356194	0.785398
4	0.5	-0.5	0.707107	0.785398	-0.7854

There are many ways around this – but end the end, you have to figure what quadrant you are in to make the value correct. You can see from the Acos values that just making the angles negative if the X,Y pair is in the 3rd or 4th quadrant would be correct – but it would introduce positive and negative values in your conversion (and make the jump from π to –π at the 2nd /3rd quadrant border.

What I do is look at the quadrant and adjust the result so all of my conversions produce a positive angle as a first pass. You still will have to return to adjust for the crossing of the 0 axis issue - which is discussed above in more detail.

Just be aware that simply using the arc cos or arc tan will give you the wrong theta value unless you adjust your result for the quadrant position.

I want to thank Bruce, Dithermaster and ddkengr for their comments. I have tried to correct my earlier output and incorporate corrections and clarifications

Ray Solomon

9-15-2019