Trigonometric Functions

okay, well there's no magic here other than using tables. That is you simply precalculate a bunch of Cosines / Sines into a lookup table and use that. In 99.99% of the time it ends up in a 256 byte long table, as we're living in a 8 bit world, and wrapping around is a nice feat.

a little pseudo c64 basic code here:

```
c=2*pi/256

for x=0 to 2*pi step c

value=(sin(x)*128)

if value<0 then value=255-value+1

poke 8192+q,sine

q=q+1

next x
```

the c64 trigonometric functions think in radian, which means that the sin/cos functions repeats themselves after 2*pi. To make that 256 steps we simply make our for/next loop advance at 2*pi/256 steps.

then as we cannot represent numbers <1 on 8 bits we simply multiply the sine value by 128. This means when we will multiply with these lookup sine values we will have to also divide by 128 to get our 'real' result. As: x*sin(y)*128=z*128. to get z we divide by 128, but that is easy as it needs only shifting bits. (lsr/ror)

Remember sin/cos functions results in numbers in the range 1>sin/cos>-1. As in c64 land we use 2's complement to represent negative numbers we have to take care of that with the “if value<0 then value=255-value+1” line. 2's complement form is also the reason for multiplying with 128, as on 8 bit a 2's complement number is in the +/- 128 range.

However there are no strict rules on how you do your sin/cos tables. You might want 16 bit values and then you have to multiply with 65535 for maximum accuracy, and change the whole precalculation process accordingly. (ie you need 2 8 bit tables with 256 entries).

Also dont forget that the cos/sin functions are almost the same: cos(x)=sin(x+2*pi/4). if you have tables with 256 entries this means: lda cos,x = lda sin+64,x. Make sure to have your sine table 64 bytes longer, and then you can save 192 bytes ;)