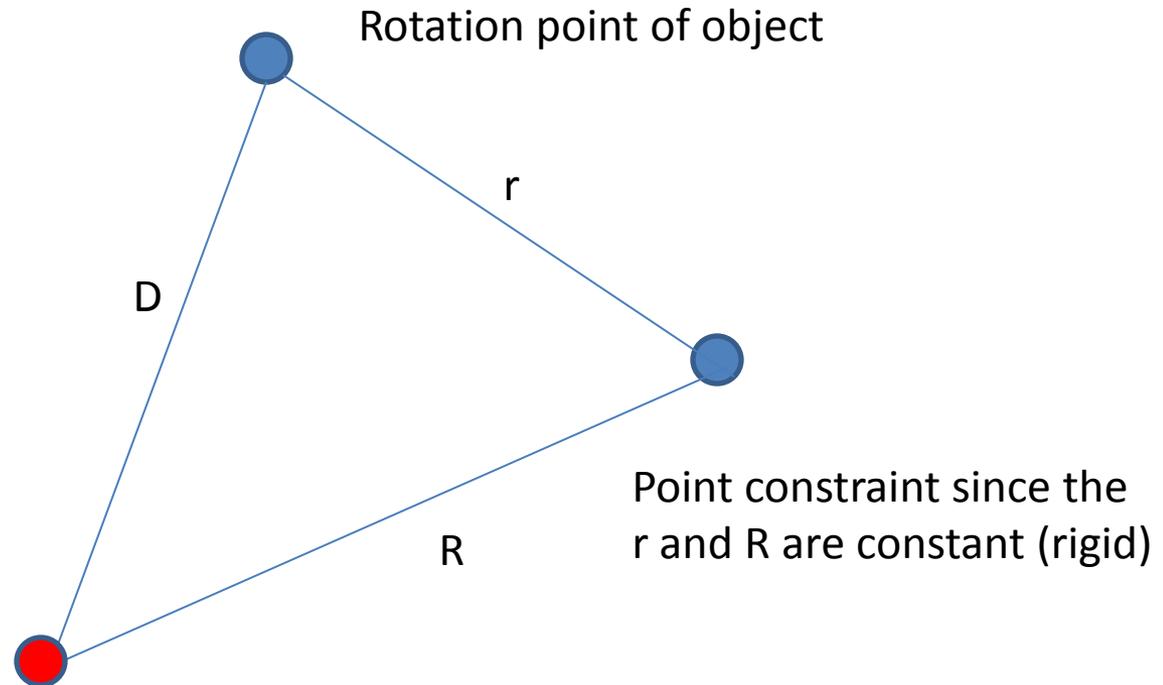
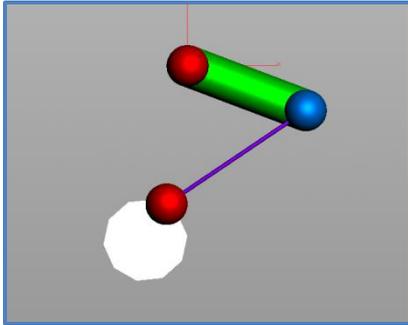


Dot product and pythagorean Theorem

by Deborah R. Fowler

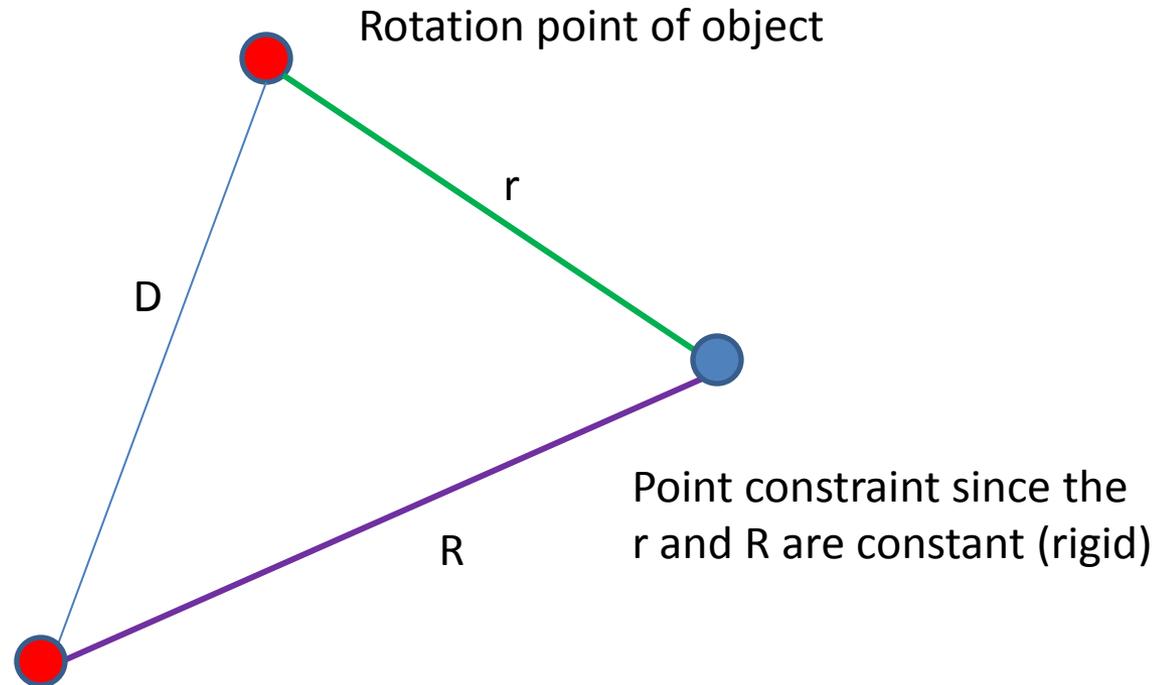
Suppose you have three points. Problem – we know the rotation point and the moving point, but we need to know the constraint point



Moving point controlling the object rotation

D changes, but we can compute it ... we would like to know the angle of rotation of the objects of r and R length

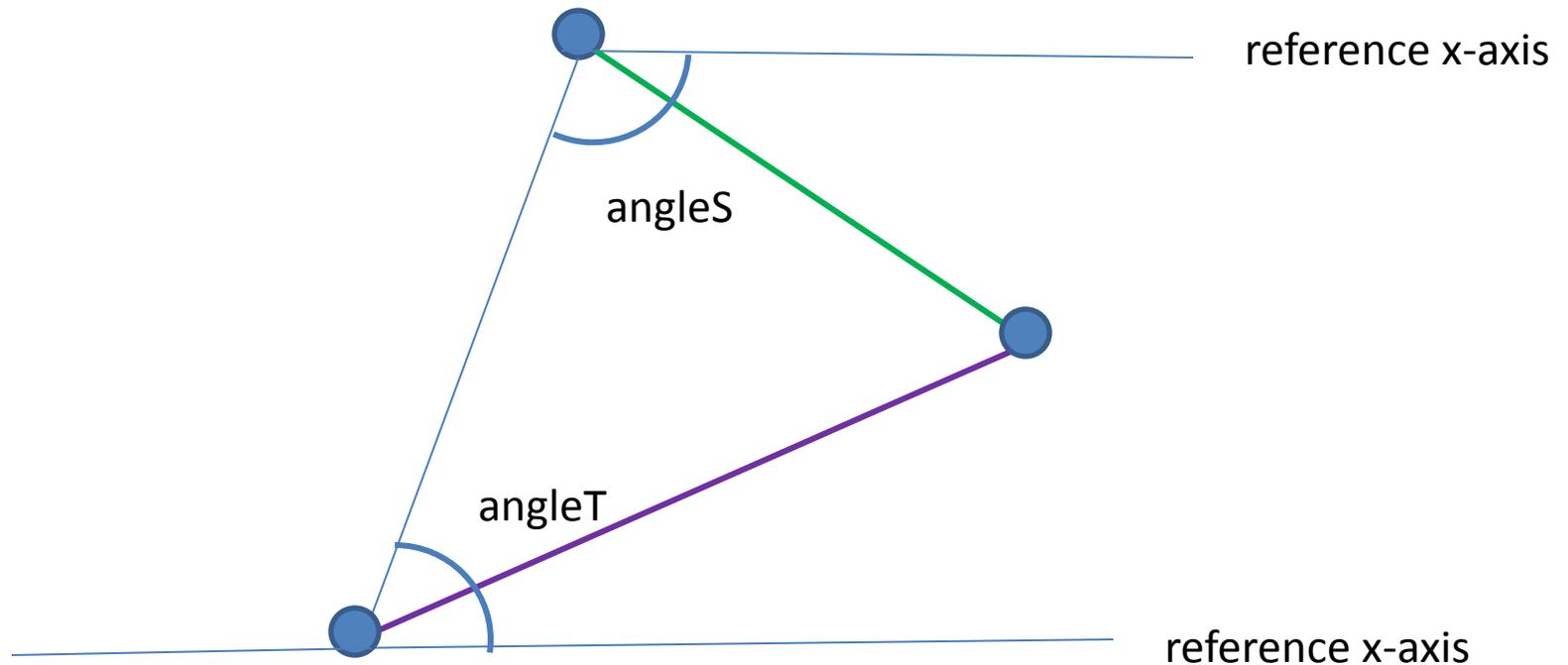
What we really want to find is how much the green(leg) and the (purple) rod rotate around their pivot points



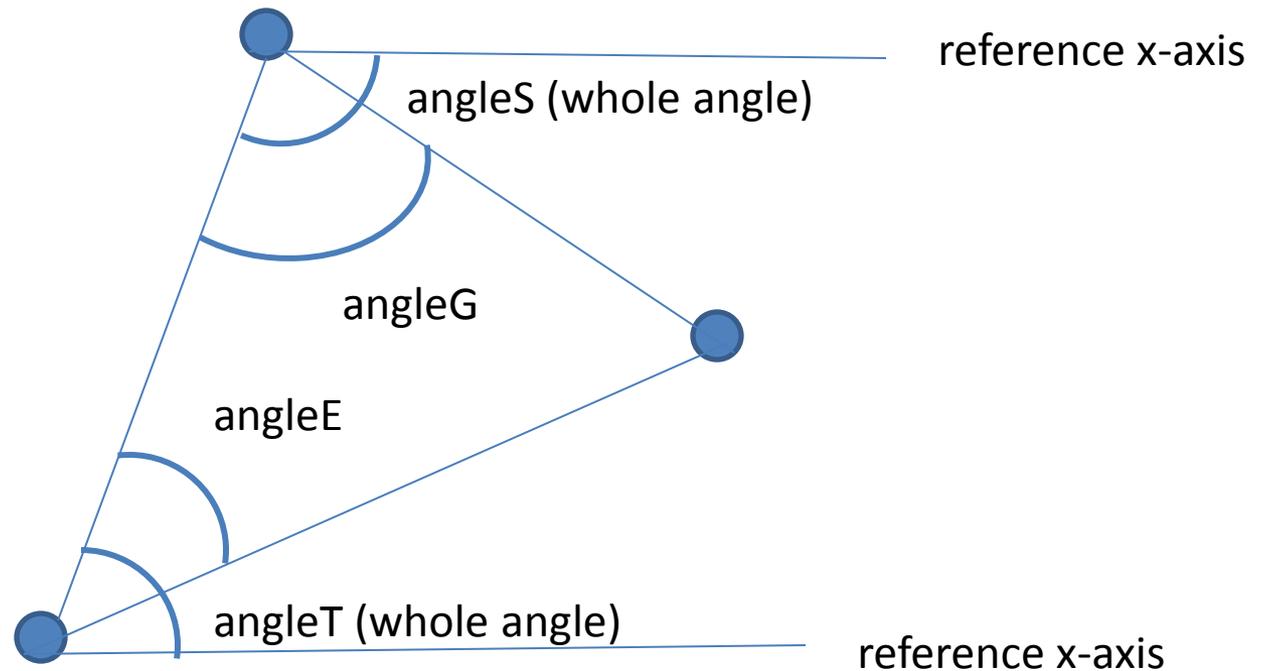
Moving point controlling the object rotation

D changes, but we can compute it ... we would like to know the angle of rotation of the objects of r and R length

By definition, **dot product** of two vectors = $\cos(\text{angle})$ * product of the length of the two vectors. We will use this property to calculate angleS and angleT , but before we do this, what else do we know?



If we can find angleT and angleS that's a start, but what we really want is angleT-angleE and angleS-angleG



We can find angleE and angleG given the **Pythagorean** theorem
That is, given a right-sided triangle, the sum of the two sides
each squared is equal to the hypotenuse squared ($a^2+b^2=c^2$)

From **Pythagorean's theorem** we know for right triangles that the two sides, each squared, equal to the hypotenuse.

$$red^2 + d^2 = R^2 \text{ and } red^2 + (D-d)^2 = r^2$$

Equating the two:

$$red^2 = R^2 - d^2 \text{ and } red^2 = r^2 - (D-d)^2$$

$$R^2 - d^2 = r^2 - (D-d)^2$$

Rewritten:

$$R^2 - d^2 - r^2 + (D-d)^2 = 0$$

$$R^2 - d^2 - r^2 + D^2 - 2Dd + d^2 = 0$$

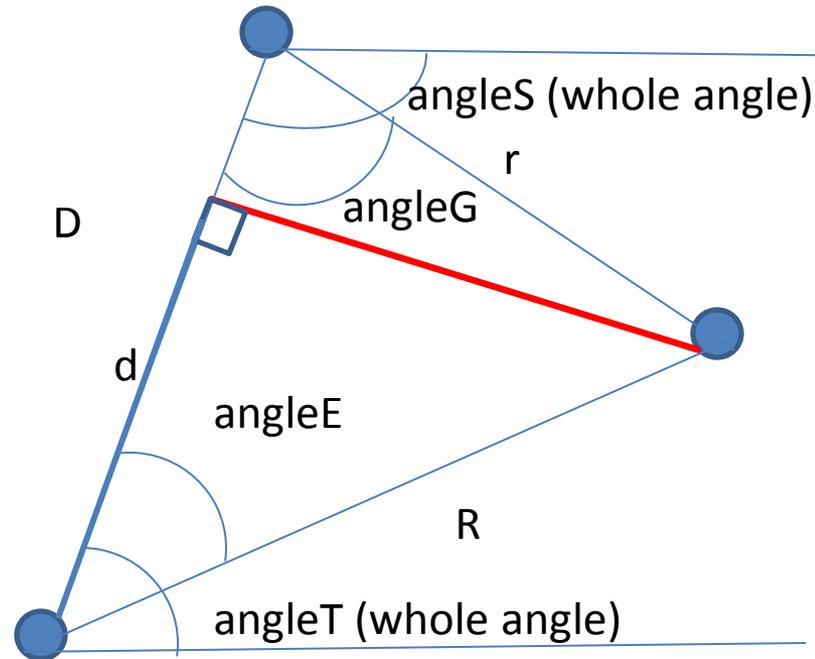
Solving for d:

$$R^2 - \cancel{d^2} - r^2 + D^2 - 2Dd + \cancel{d^2} = 0$$

$$R^2 - r^2 + D^2 = 2Dd$$

$$d = (R^2 - r^2 + D^2) / 2D$$

So what does this give us?



We can calculate length D at any given moment since we know the point information. We also know r and R's length – they are given.

By the definition of cosine, we can find angleE and angleG.

So what do we have so far?

$$d = (R^2 - r^2 + D^2) / 2D$$

Recall the definition of cosine

cos = adjacent/hypotenuse

$$\cos(\text{angleE}) = d / R$$

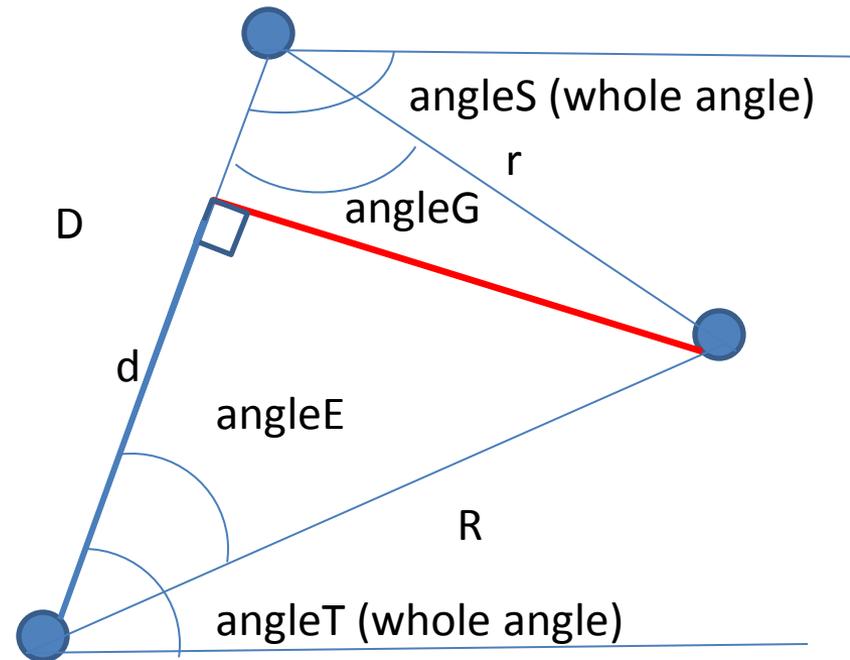
$$\cos(\text{angleG}) = (D-d)/r$$

Thus taking the inverse cosine

$$\text{angleE} = \cos^{-1} (d/R)$$

$$\text{angleG} = \cos^{-1} ((D-d)/r)$$

\cos^{-1} is also called inverse cosine or the arc cosine and is available as **acos** in many packages



So we have a way to compute angleE and angleG. We said we could compute angleS and angle T from the dot product. Let's do that now:

The dot product is defined to be:

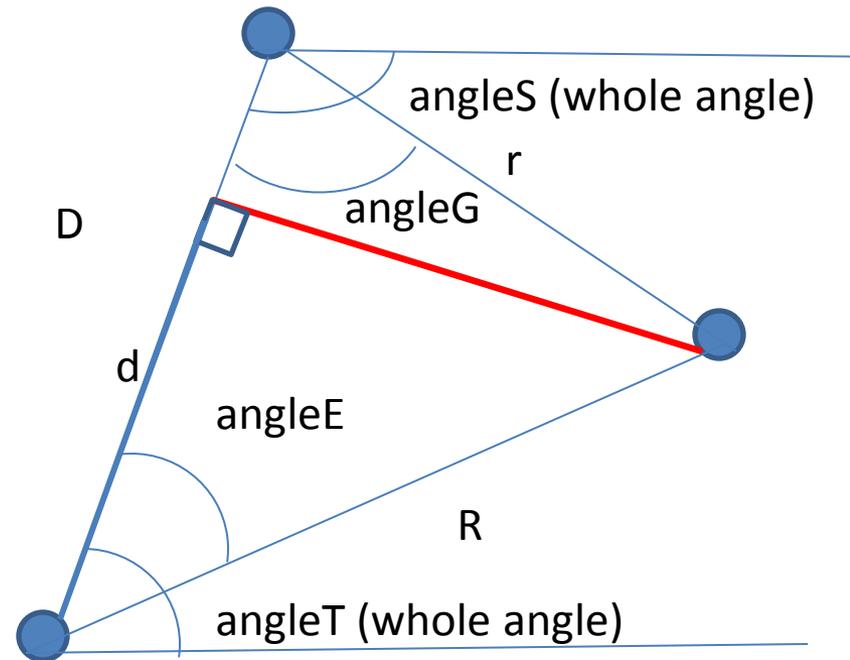
$$A \cdot B = |A| * |B| * \cos(\text{angle})$$

If the vectors are normalized, then it is simply
 $\text{normalized}A \cdot \text{normalized}B = \cos(\text{angle})$
(ie. $|A|$ and $|B|$ are one)

Given A is (A_x, A_y) and B is (B_x, B_y)

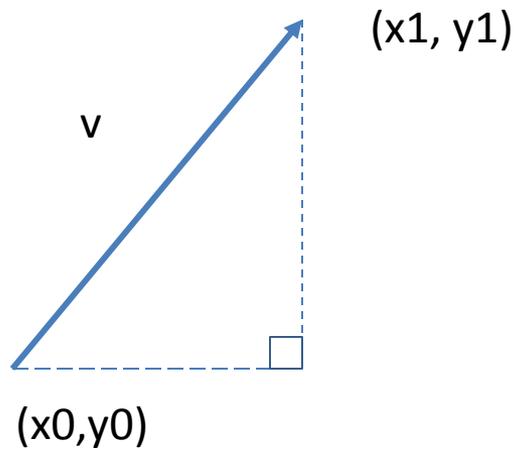
$$A \cdot B = A_x * B_x + A_y * B_y$$

So how do we get the vector D?

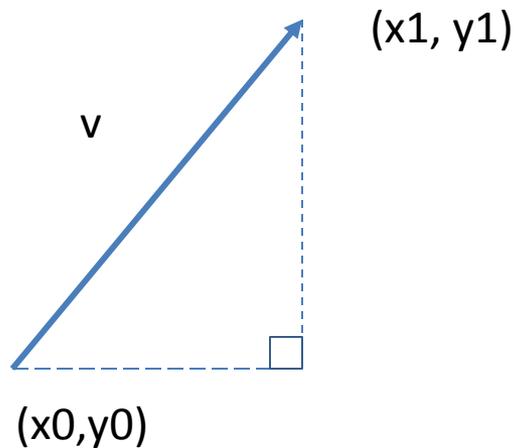


A great explanation on dot product can be found at
<http://www.mathsisfun.com/algebra/vectors-dot-product.html>

Vectors – given two points in space, the vector in the diagram is defined by $(x_1 - x_0, y_1 - y_0)$



Vectors: Recall how to compute the length of a vector and how to normalize a vector



Given a vector v
The length of a vector is
 $\text{sqrt}((x_1 - x_0)^2 + (y_1 - y_0)^2)$

This is the length from
pythagorean's theorem

To normalize a vector, we divide it
by the length of the vector

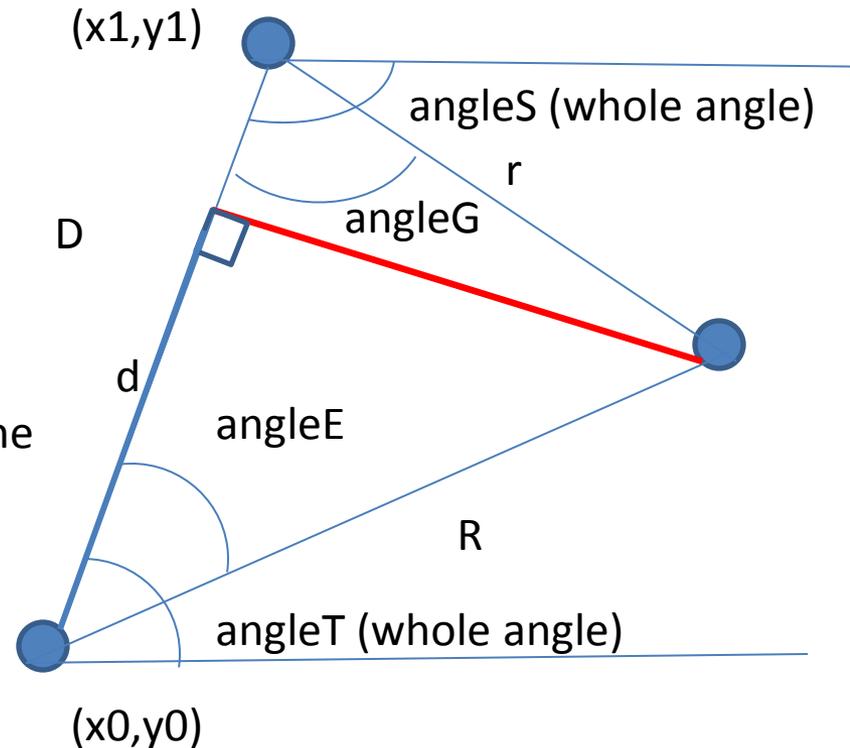
What about our case?

Recall that if A is (A_x, A_y) and B is (B_x, B_y)
normalizedA · normalizedB = $\cos(\text{angle})$
and
 $A \cdot B = A_x * B_x + A_y * B_y$

Well, one of our vectors is the reference x-axis, the
other vecD is represented by (x_1, x_0, y_1, y_0)
therefore the dot product is:

$$(1,0) \cdot (x_1 - x_0, y_1 - y_0)$$
$$1 * x_1 - x_0 + 0 * y_1 - y_0$$

Which is $x_1 - x_0$. If we normalize this we end up with
 $\cos(\text{angleT}) = (x_1 - x_0) / (\text{length of vecD})$
 $\text{angleT} = \arccos((x_1 - x_0) / \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2})$
 $\text{angleS} = 180 - \text{angleT}$



So we are almost done:

$$\text{angleT} = \text{acos}((x1-x0) / \text{sqrt}((x1-x0)^2 + (y1-y0)^2))$$

$$\text{angleS} = 180 - \text{angleT}$$

$$\text{angleE} = \text{acos}(d/R)$$

$$\text{angleG} = \text{acos}((D-d)/r)$$

$$d = (R^2 - r^2 + D^2) / 2D$$

We already know r and R

$$D = \text{sqrt}((x1-x0)^2 + (y1-y0)^2)$$

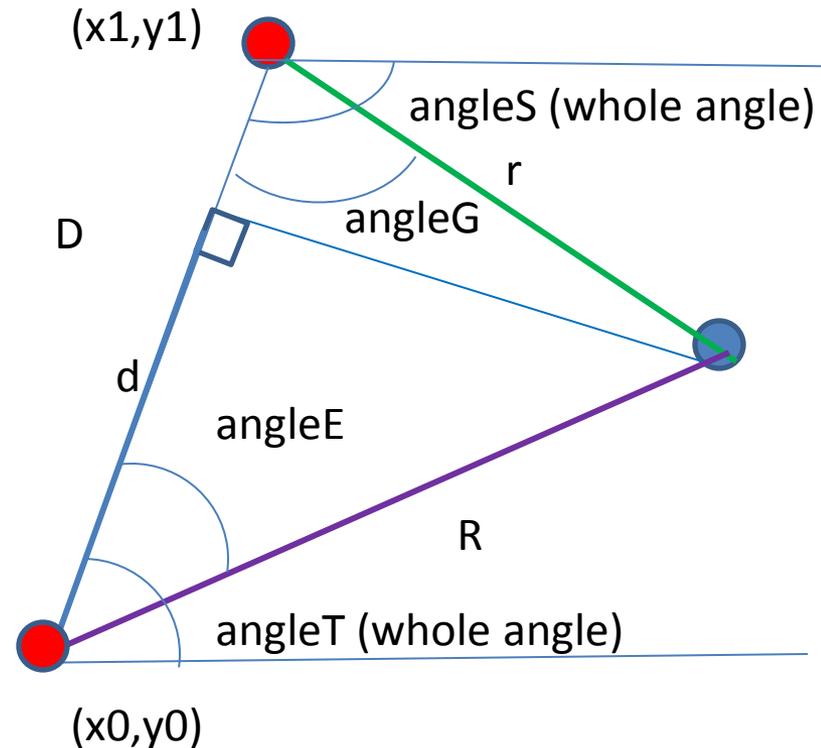
So we have all the information we need to

Give us the rotations which are

$\text{angleT} - \text{angleE}$ for our purple object

$\text{angleG} - \text{angleS}$ for our green object ie. $-(\text{angleS} - \text{angleG})$

If you want to see a real application of this, check out the sample file.



In the sample file, hscript looks like this:

Using multi-line expressions, we have for the rotate on z variable of the [purple object](#)

```
{
# Expression calculating the angle of rotation, from the diagrams this is angleT - angleE
# D is the distance between the two centers of the circles
#  $D = \sqrt{(R^2 - r^2 + D^2) / 2D}$  where D is the distance between the points
# for the moment lets assume the x1, y1 is at the origin
#
R = .4;
r = .3;
x1 = 0;
y1 = 0;
x0 = point("../xformRotatingWheel",40,"P",0);
y0 = point("../xformRotatingWheel",40,"P",1);
D = sqrt(pow(x1-x0,2) + pow(y1-y0,2));
d = (R*R - r*r + D*D)/(2.0*D);
angleE = acos(d/R);

# next compute angleT
angleT = acos( ( x1 - x0 )/D);
angleRot = angleT - angleE;
return angleRot;
}
```

In the sample file, hscript looks like this:

Using multi-line expressions, we have for the rotate on z variable of the **green object**

```
{
# Expression calculating the angle of rotation, from the diagrams this is angleS - angleG
# D is the distance between the two centers of the circles
#  $D = \sqrt{(R^2 - r^2 + D^2) / 2D}$  where D is the distance between the points
# for the moment lets assume the x1, y1 is at the origin, but the equations are still valid if you adjust this
#
R = .4;
r = .3;
x1 = 0;
y1 = 0;
x0 = point("../xformRotatingWheel",40,"P",0);
y0 = point("../xformRotatingWheel",40,"P",1);
D = sqrt(pow(x1-x0,2) + pow(y1-y0,2));
d = (R*R - r*r + D*D)/(2.0*D);
angleG = acos((D-d)/r);

# next compute angleT
angleT = acos( ( x1 - x0 )/D);
angleS = 180 - angleT;

angleRot = angleS - angleG;
return -angleRot;
}
```

in the sample file,
`dotPythagoreanInAction.hipnc`

- see the red nodes for the equations
- the yellow node is where the rotation of the point (such as a gear that will drive the animation) is located
- note that the x-axis is the reference axis

in the sample file,
dotPythagoreanInAction.hipnc

