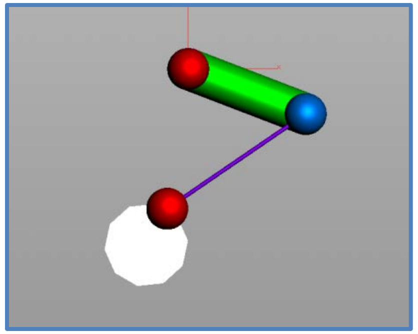
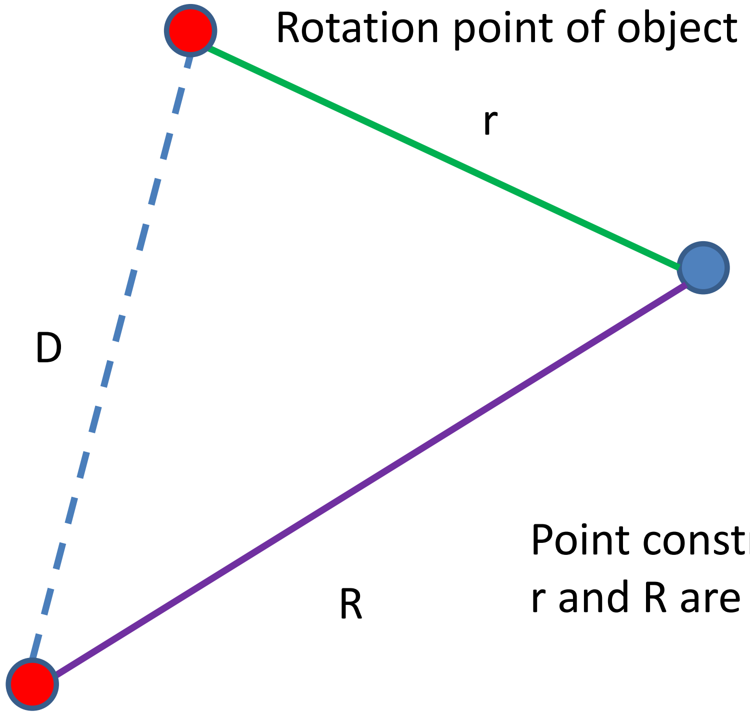
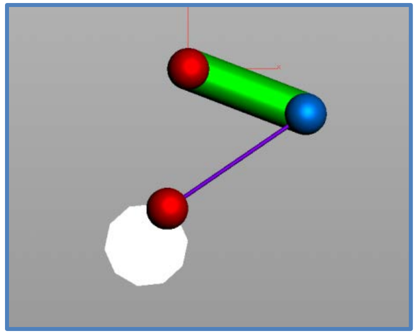
Dot product and Pythagorean Theorem

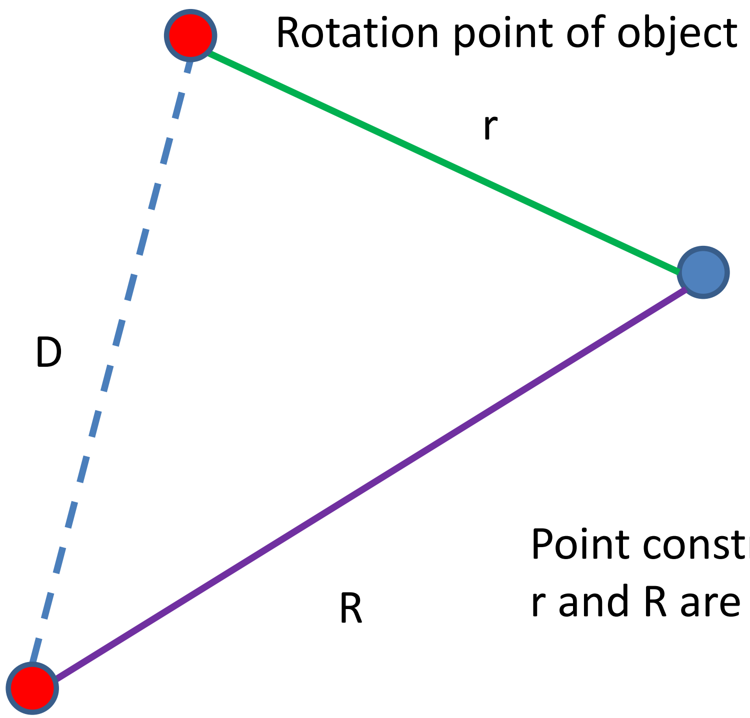
by Deborah R. Fowler

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





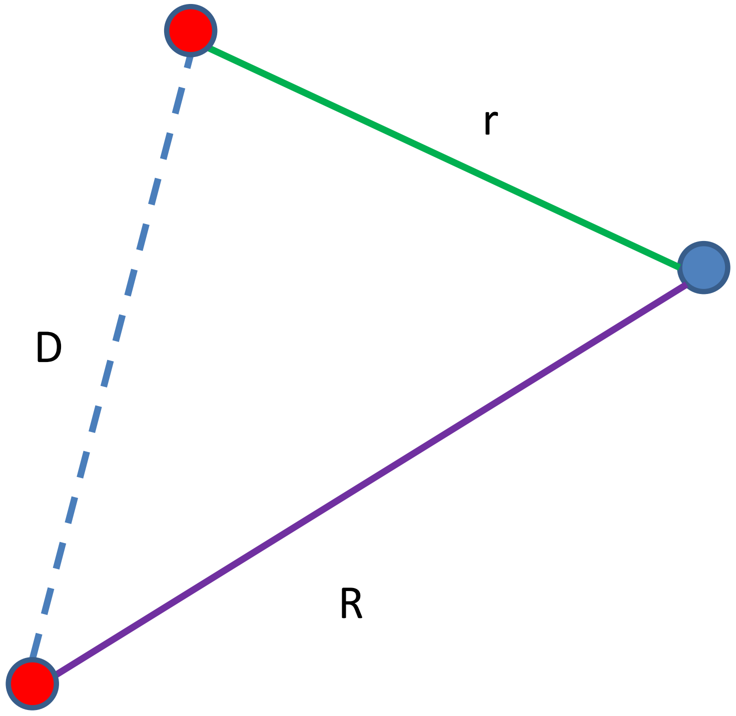




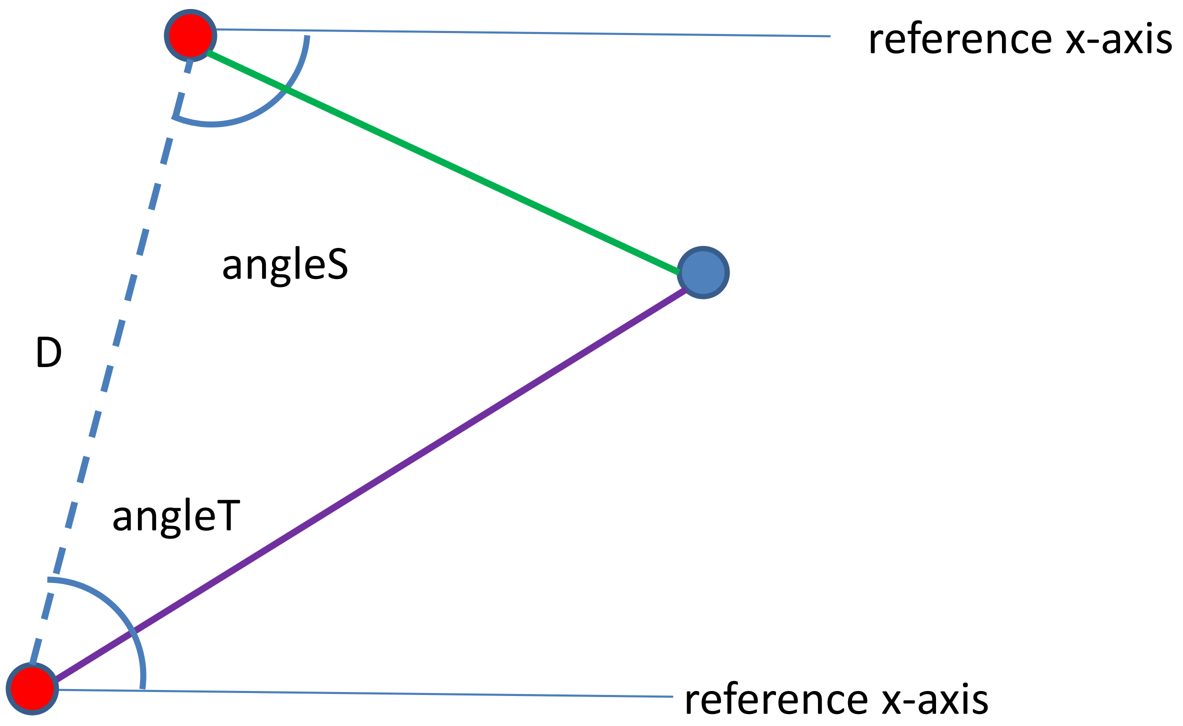
Point constraint since the
r and R are constant (rigid)

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

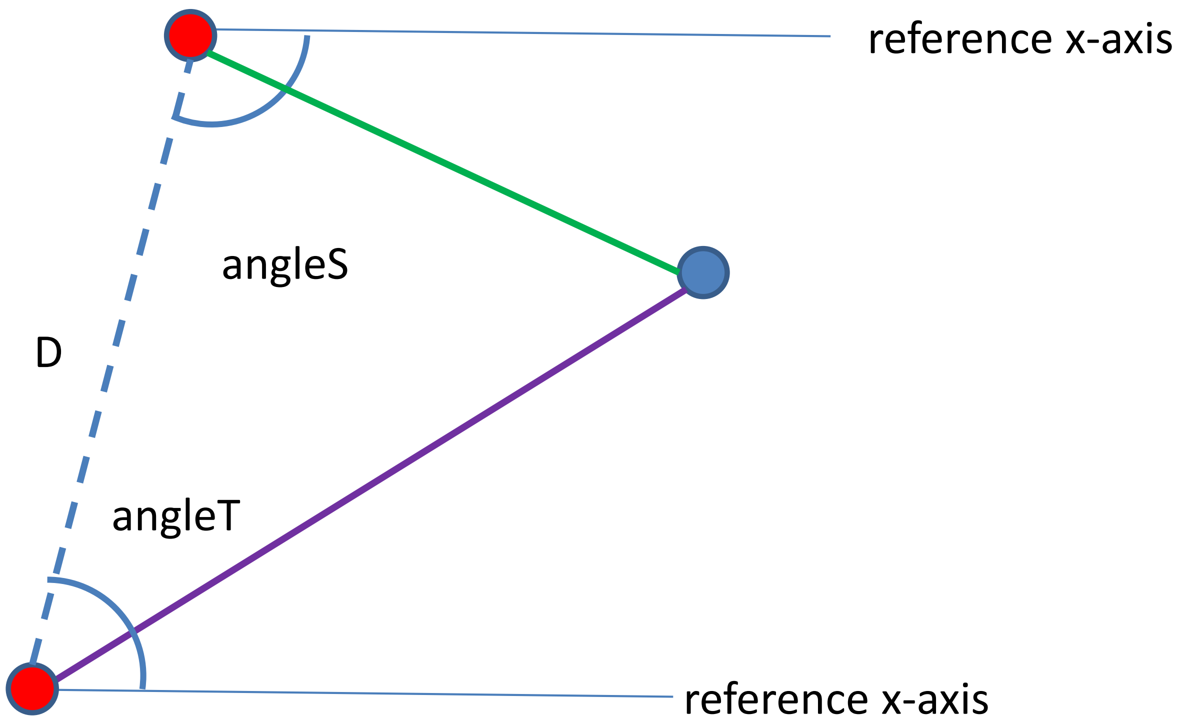
What we really want to find is how much the green and the (purple) rods rotate around their pivot points



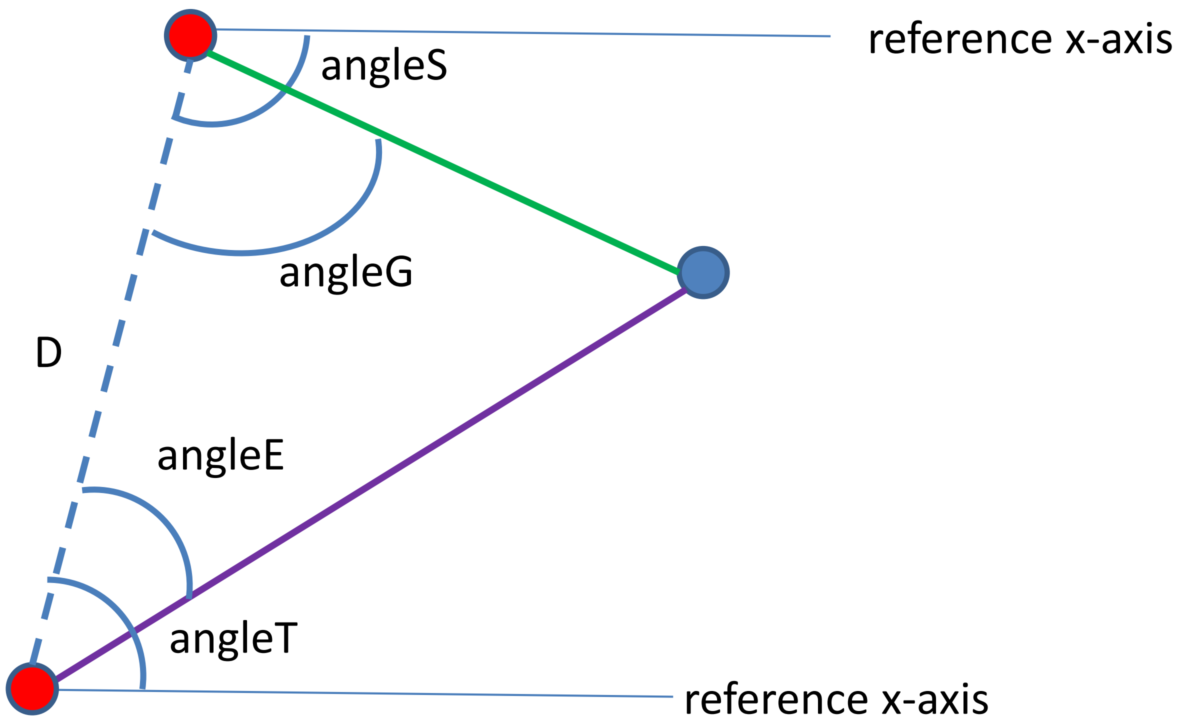
By definition, **dot product** of two vectors = cos (angle) * product of the length of the two vectors. We will use this property to calculate angleS and angleT, but before we do this, what else do we know?



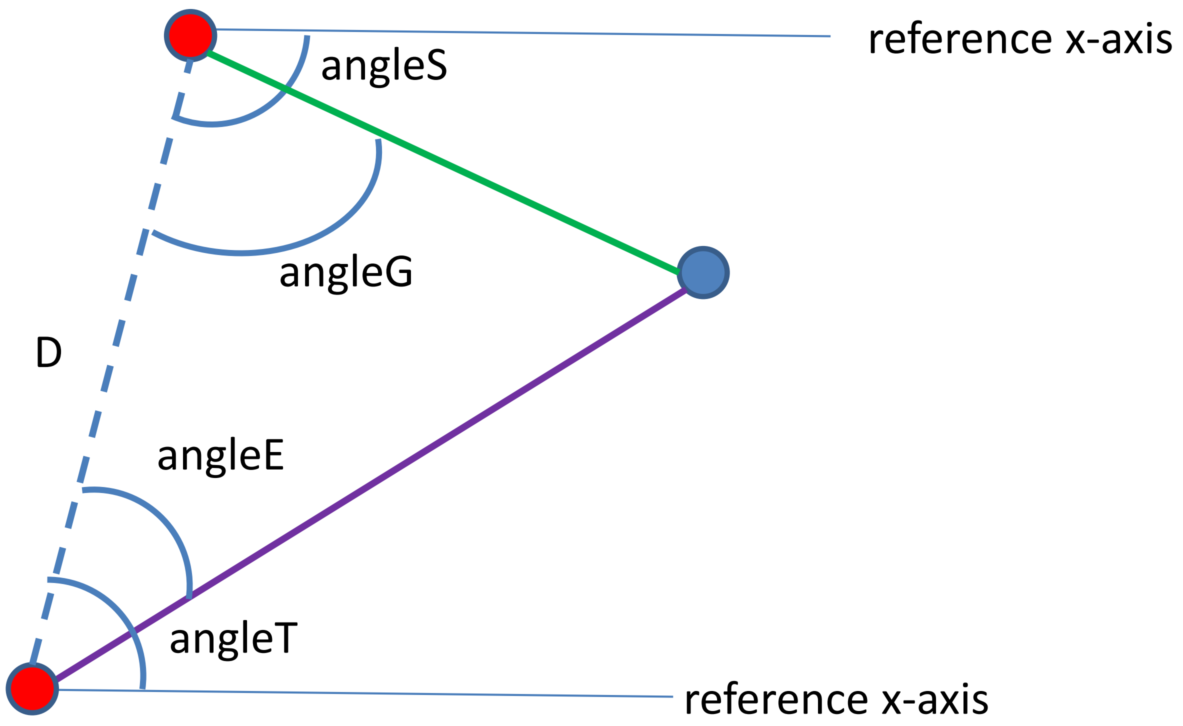
If we can find angleT and angleS that's a start …



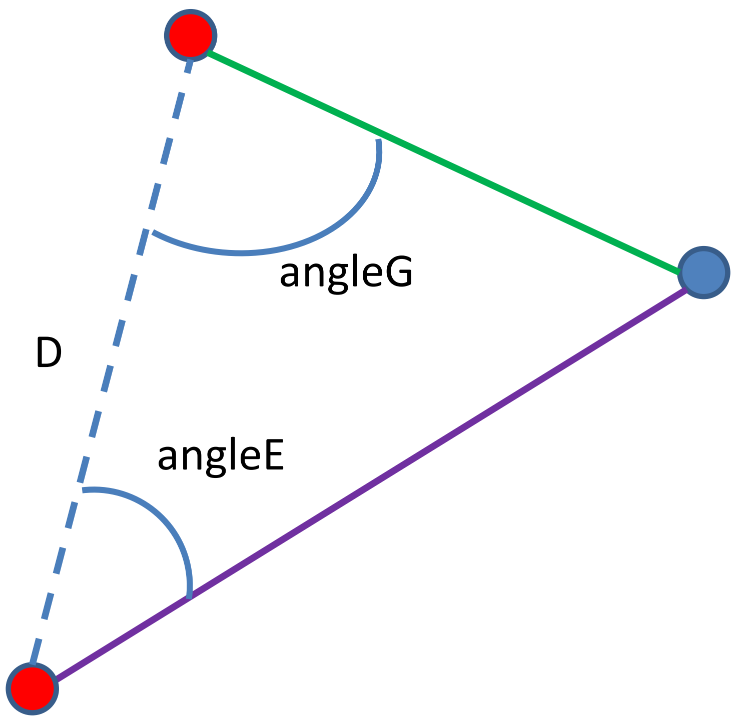
… but what we really want is angleT-angleE and angleS-angleG



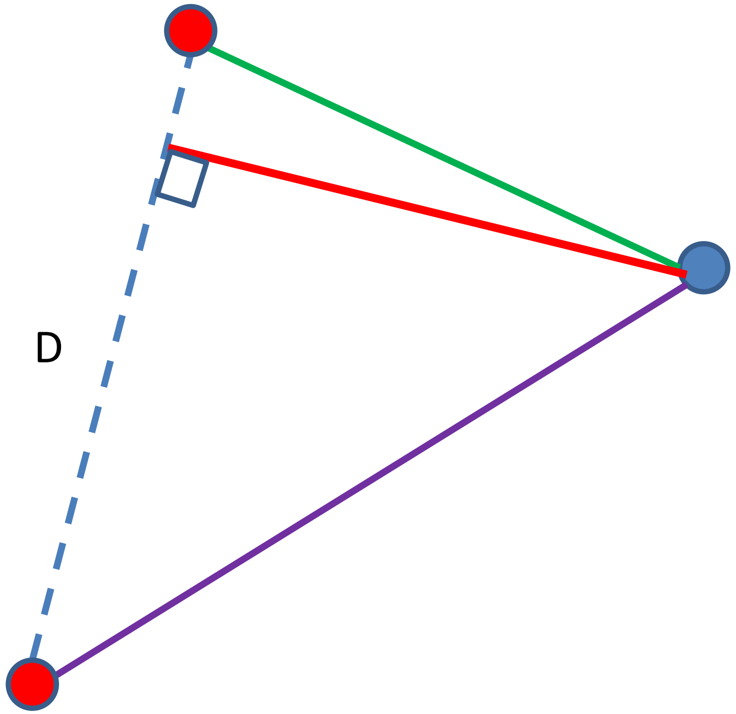
Pythagorean theorem can help us find angleG and angleE



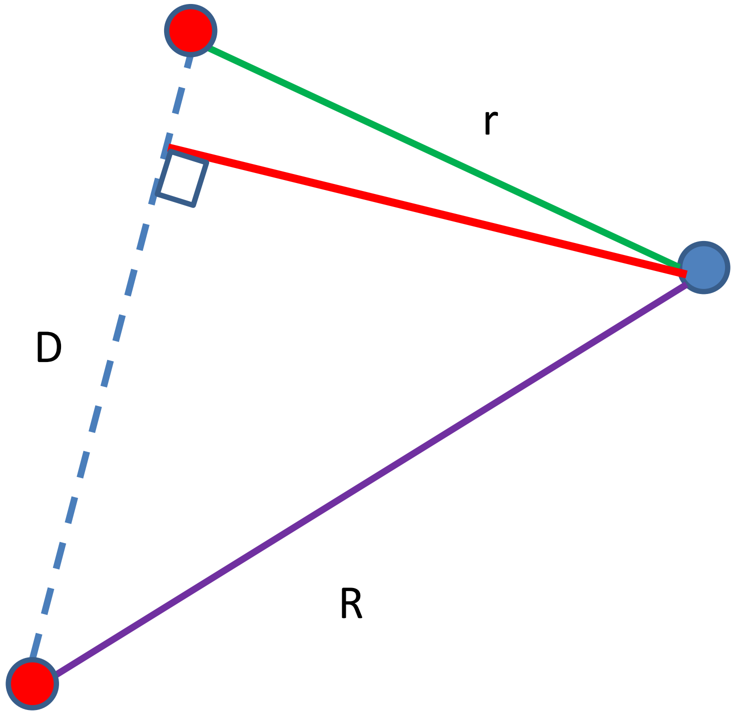
But didn’t Pythagorean only apply to right angled triangles?

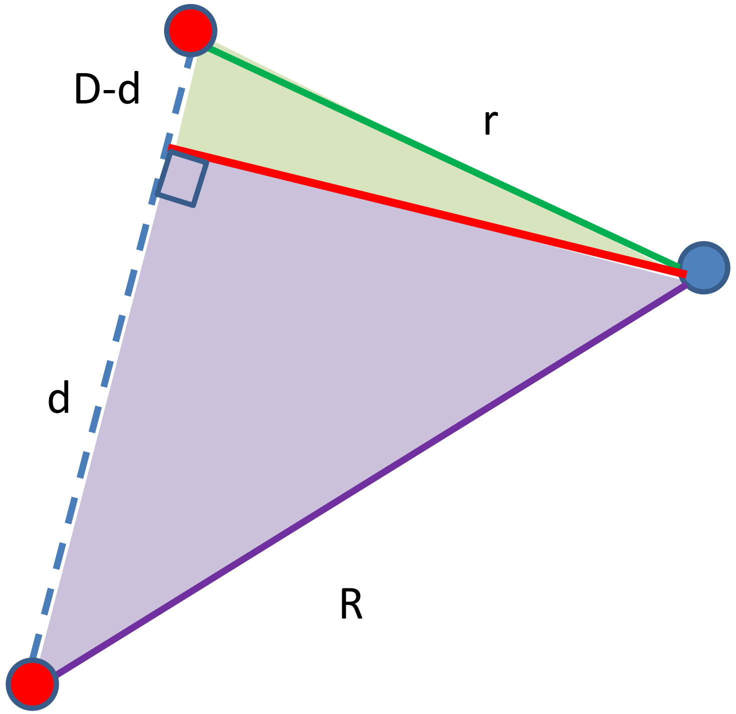


So let’s break it down … drop down a perpendicular line

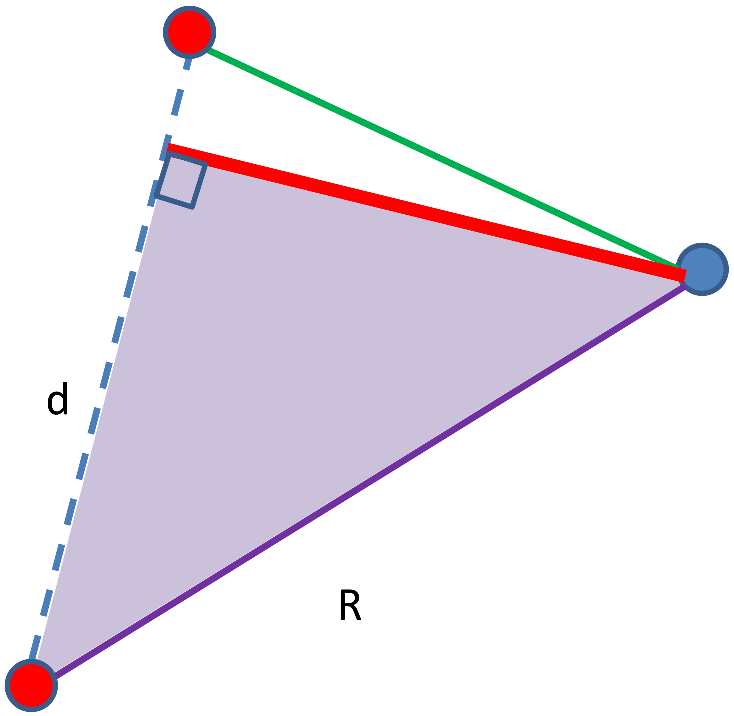


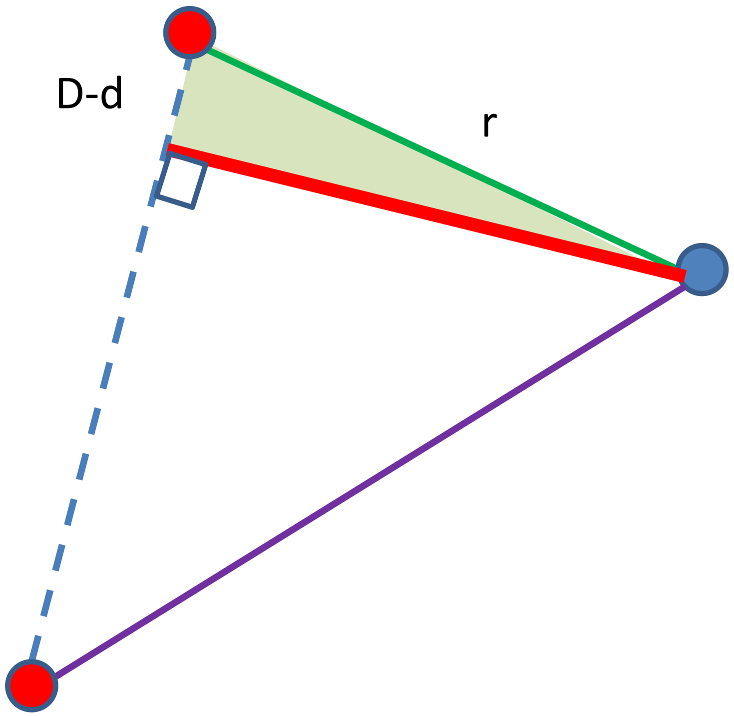
From **Pythagorean's theorem** we know for right triangles that the sum of the two sides, each squared, equals the hypotenuse squared




D-d
r
d
R

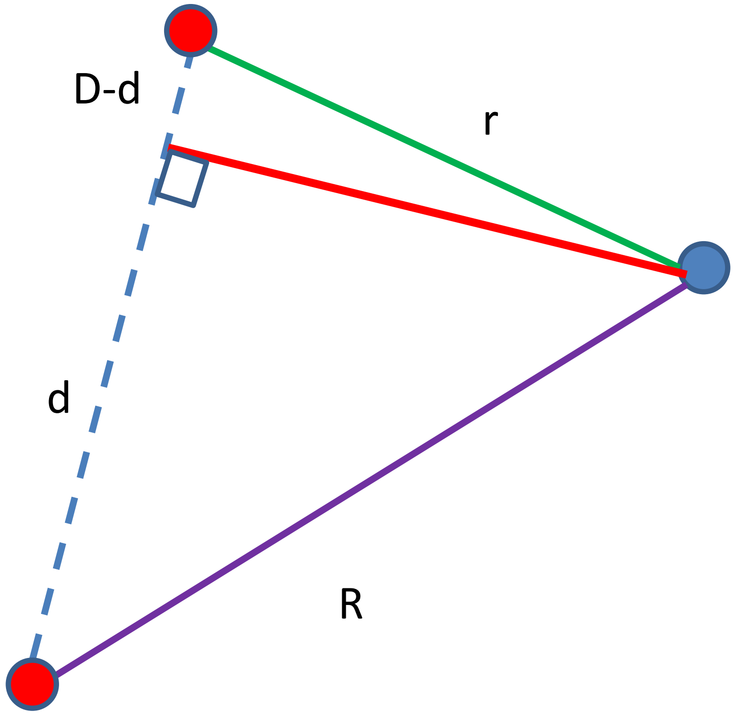
$$red^2 + d^2 = R^2 \quad \text{and} \quad red^2 + (D-d)^2 = r^2$$





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

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



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

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

Equating the two:

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

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

Rewritten:

$R^2 - d^2 - r^2 + (D\text{-}d)^2 = 0$

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

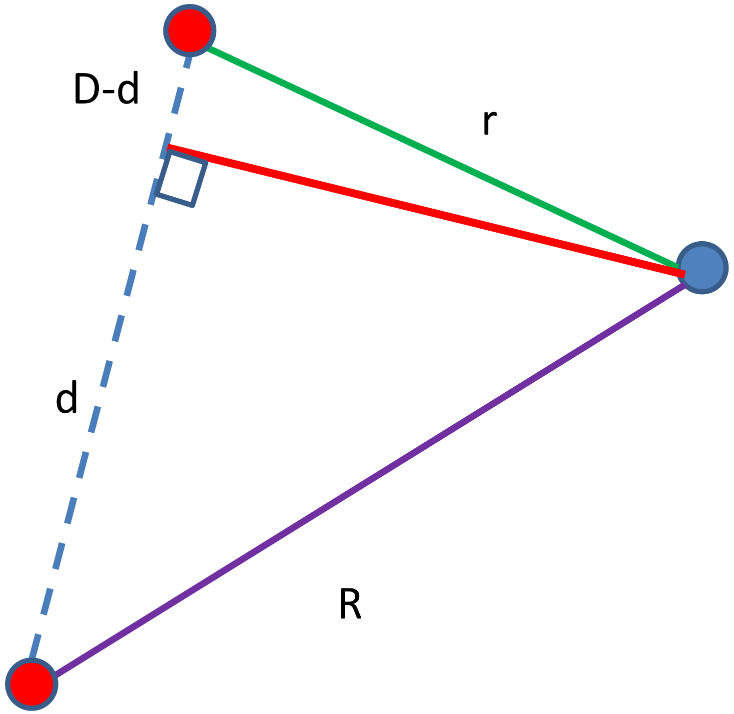
Solving for d:

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

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

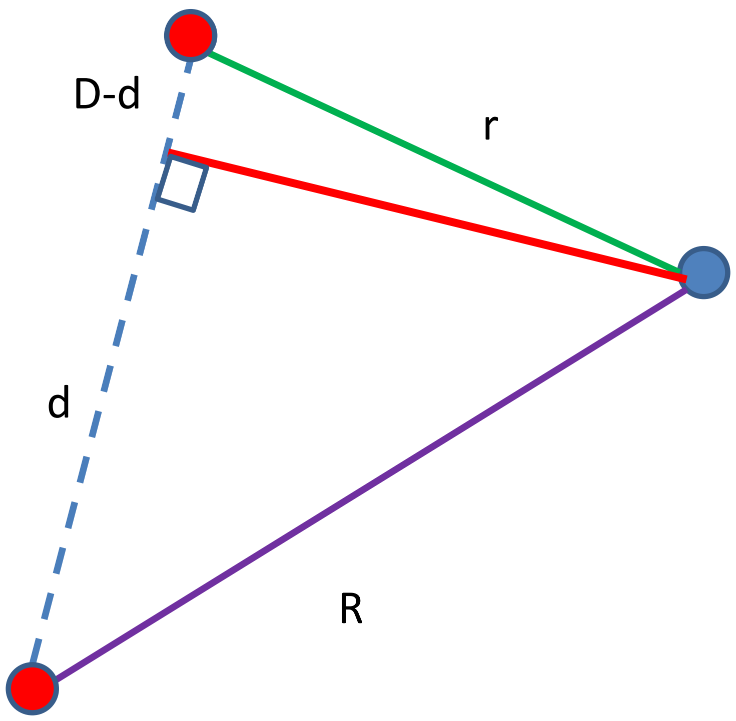
$d = (R^2 - r^2 + D^2) / 2D$

So what does this give us?



$d = (R^2 - r^2 + D^2) / 2D$ and we know r, R

We can calculate D since we know the points at any given moment



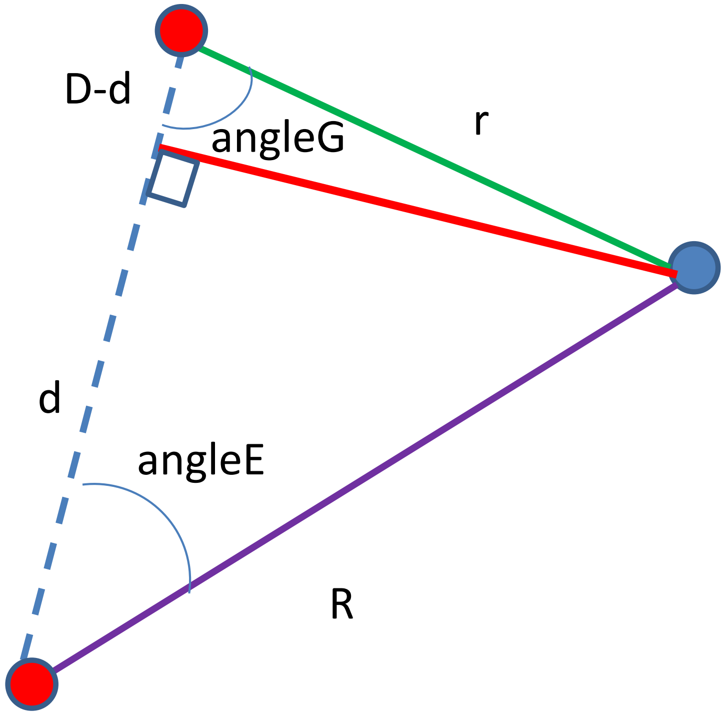
By the definition of cosine, we can find angleE and angleG.

Recall the definition of cosine
cos = adjacent/hypothenus

cos(angleE) = d / R
cos(angleG) = (D-d)/r

Thus taking the inverse cosine
angleE = $\cos^{-1}$ (d/R)
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



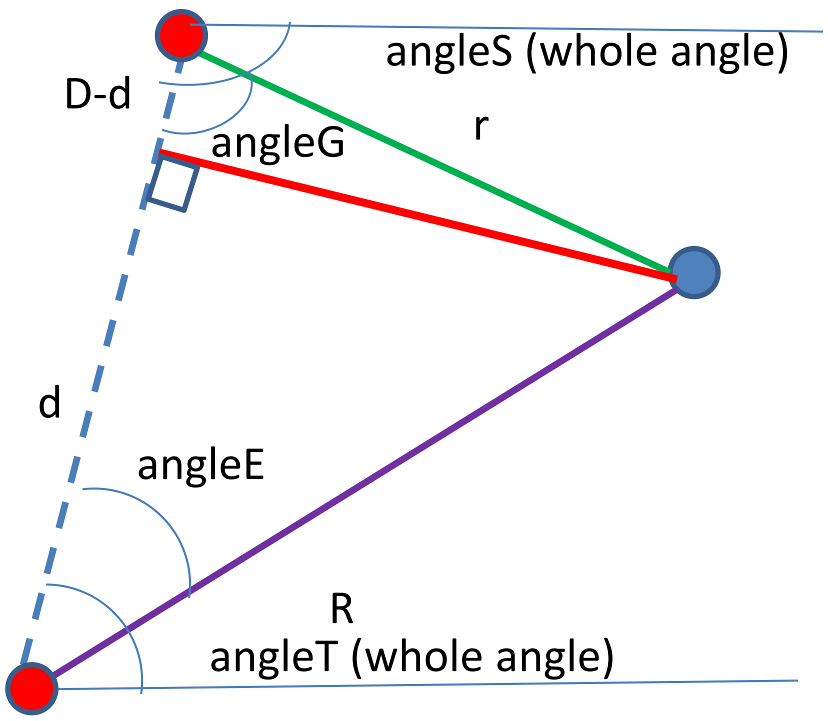
So we have a way to compute angleE and angleG. We said we could compute angleS and angle T from the dot product. Let's do that now:

The dot product is defined to be:

A · B = |A| * |B| * cos(angle)
If the vectors are normalized, then it is simply
normalizedA · normalizedB = cos(angle)
(ie. |A| and |B| are one)

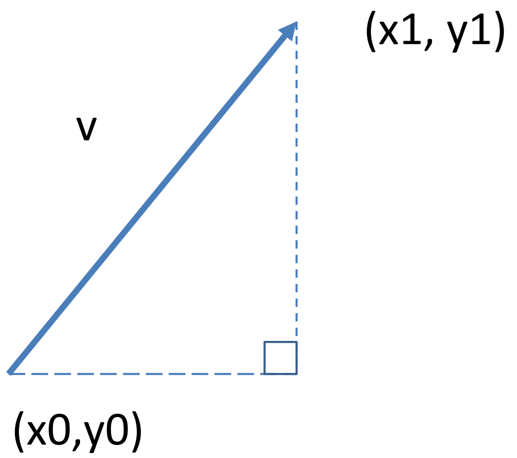
Given A is (Ax,Ay) and B is (Bx,By)
A · B = Ax * Bx + Ay * By

So how do we get the vector D?

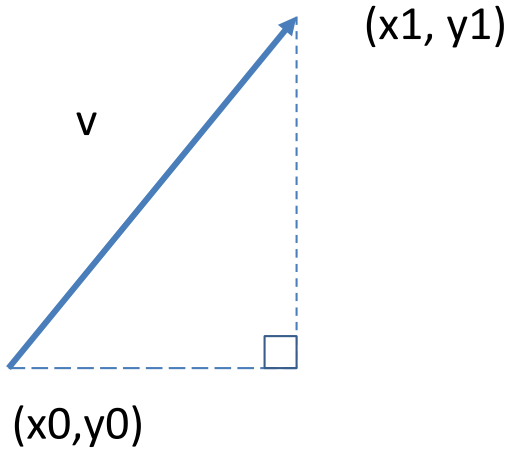


A great explanation on dot product can be found at http://www.mathsisfun.com/algebra/vectors-dot-product.html

Vectors – given two points in space, the vector in the diagram is defined by (x1-x0, y1-y0)



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

This is the length from pythagorean's theorem

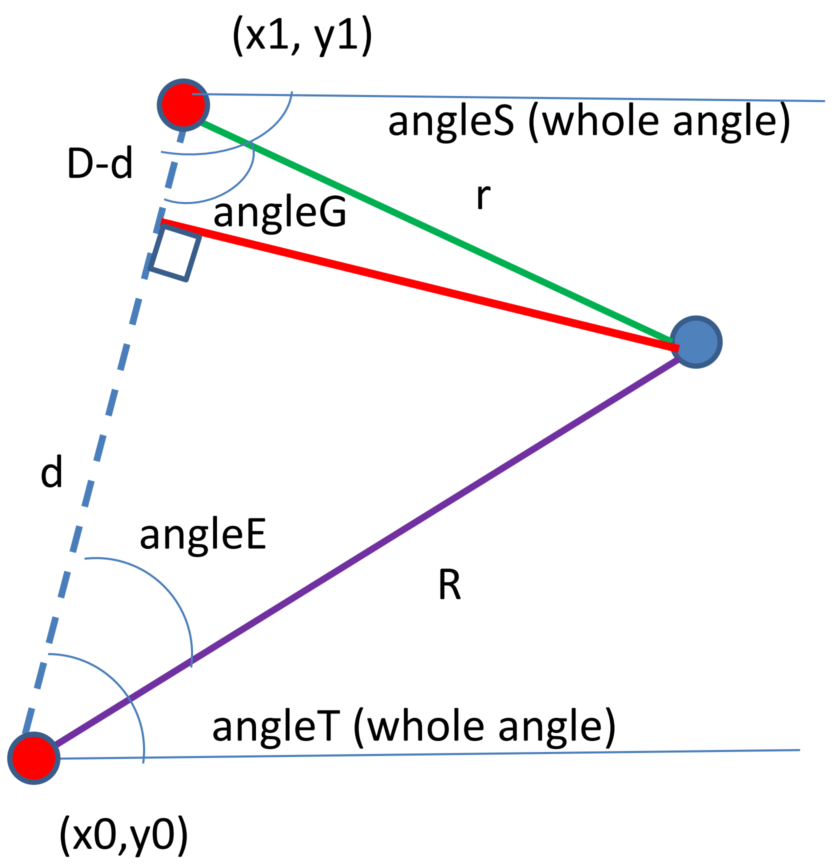
To normalize a vector, we divide it by the length of the vector

What about our case?

Recall that if A is (Ax,Ay) and B is (Bx,By)
normalizedA · normalizedB = cos(angle)
and
A · B = Ax * Bx + Ay * By

Well, one of our vectors is the reference x-axis, the other vecD is represented by (x1,x0, y1,y0) therefore the dot product is:

(1,0) · (x1-x0, y1-y0)
1 * x1-x0 + 0 * y1-y0
Which is x1-x0. If we normalize this we end up with
cos(angleT) = x1-x0 / (length of vecD)
angleT = acos((x1-x0)/ sqrt((x1-x0) ^{2} + (y1-y0)^2)
angleS = 180 - angleT



So we are almost done:

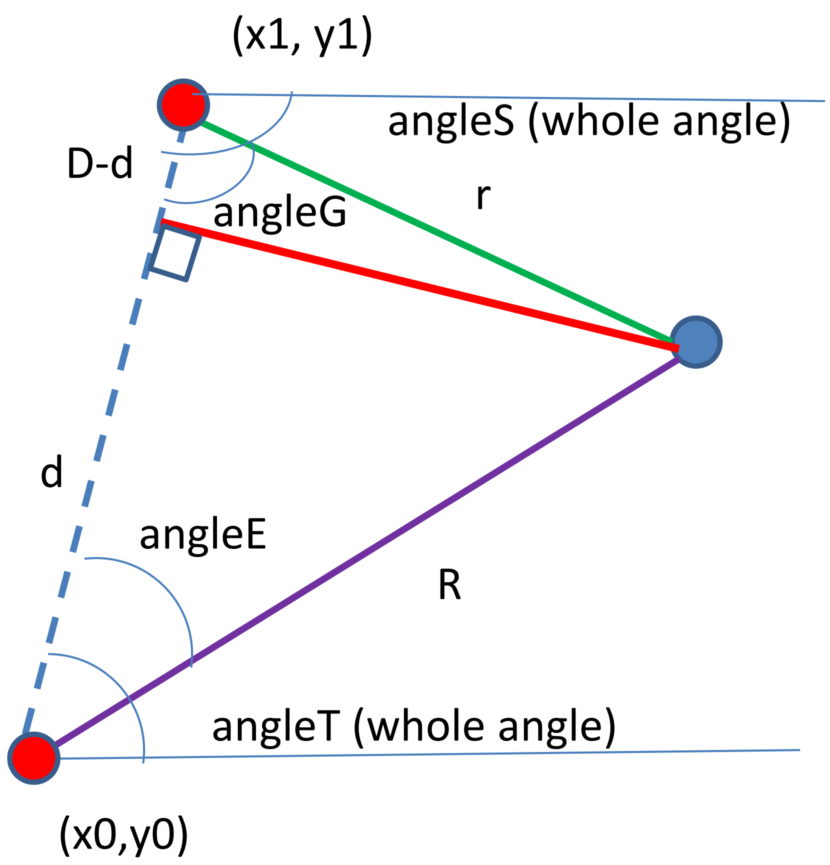
angleT = acos((x1-x0)/ sqrt($(x1-x0)^2$ + $(y1-y0)^2$)
angleS = 180 – angleT

angleE = acos (d/R)
angleG = acos ((D-d)/r)

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

We already know r and R
D = sqrt($(x1-x0)^2$ + $(y1-y0)^2$)

So we have all the information we need to
Give us the rotations which are
angleT – angleE for our purple object
angleG – angleS for our green object ie. –(angleS-angleG)



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 = ( R squared - r squared + D squared)/ 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:

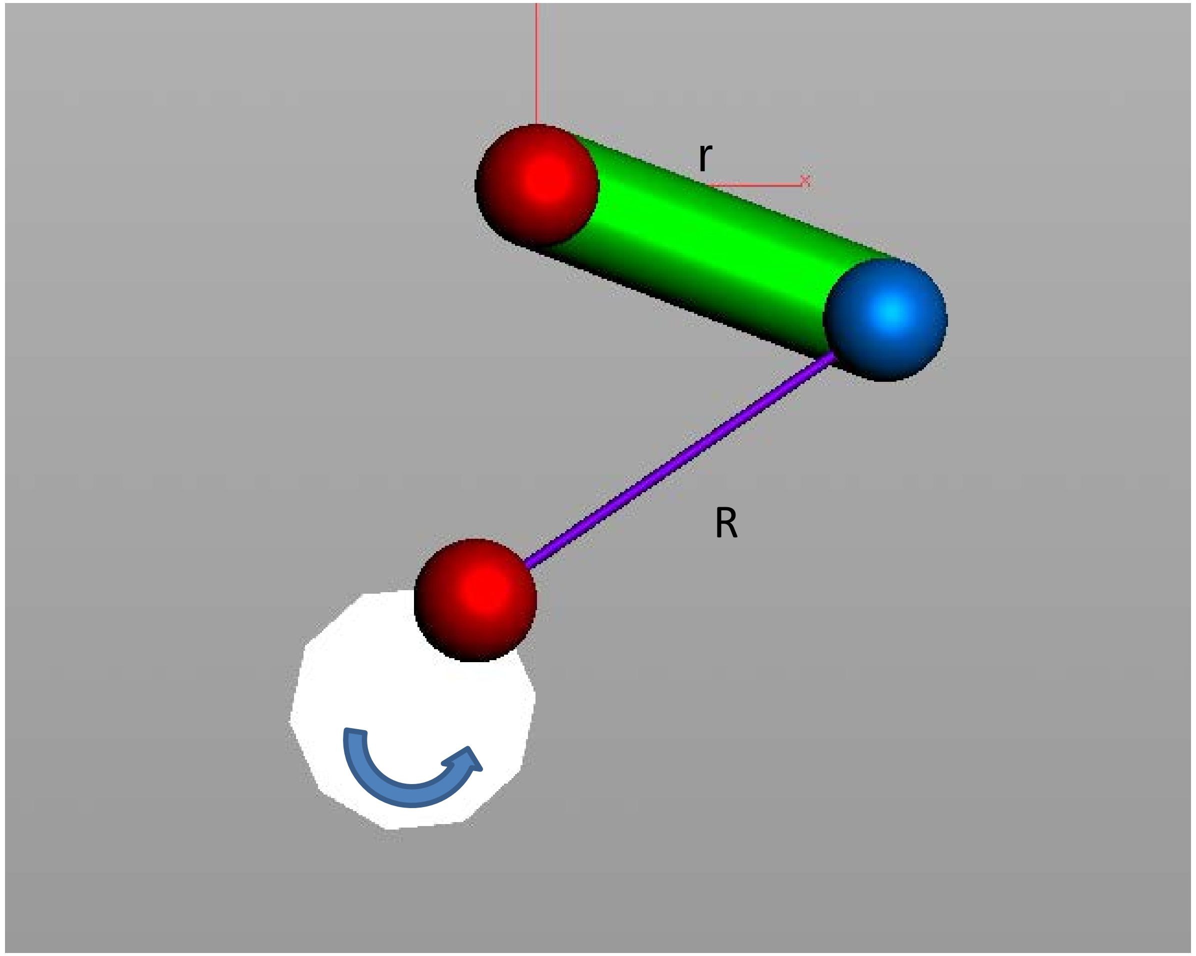
Using multi-line expressions, we have for the rotate on z variable of the green object

```
{
#  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 = ( R squared - r squared + D squared)/ 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;
}
```

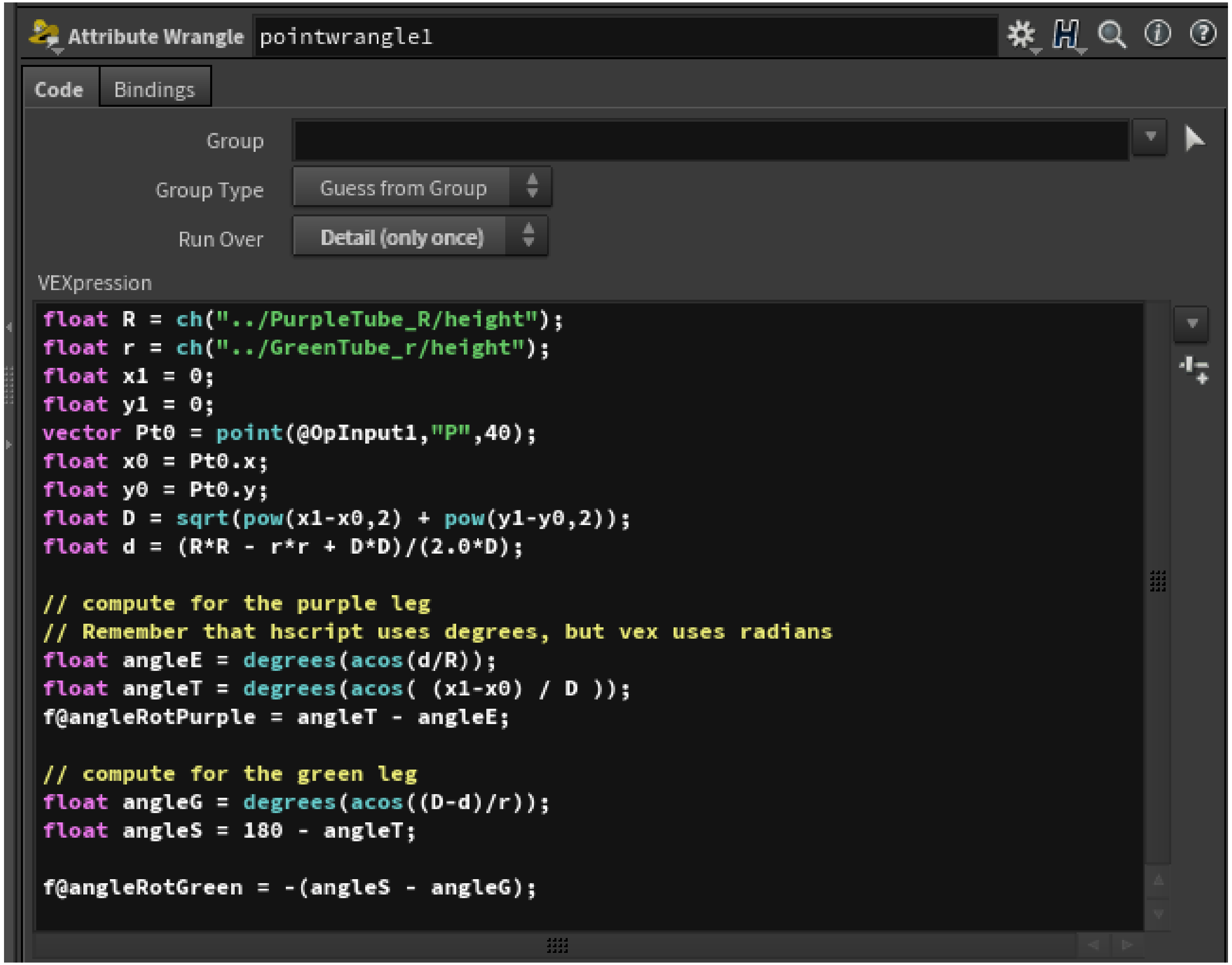
in the sample file, dotPythagoreanInAction.hipnc

- 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



Or even better … wrangle node code!




```
float R = ch("../PurpleTube_R/height");
float r = ch("../GreenTube_r/height");
float x1 = 0;
float y1 = 0;
vector Pt0 = point(@OpInput1,"P",40);
float x0 = Pt0.x;
float y0 = Pt0.y;
float D = sqrt(pow(x1-x0,2) + pow(y1-y0,2));
float d = (R*R - r*r + D*D)/(2.0*D);

// compute for the purple leg
// Remember that hscript uses degrees, but vex uses radians
float angleE = degrees(acos(d/R));
float angleT = degrees(acos( (x1-x0) / D ));
f@angleRotPurple = angleT - angleE;

// compute for the green leg
float angleG = degrees(acos((D-d)/r));
float angleS = 180 - angleT;

f@angleRotGreen = -(angleS - angleG);
```

To access detail attributes in transforms

detail(path,attribute,which)