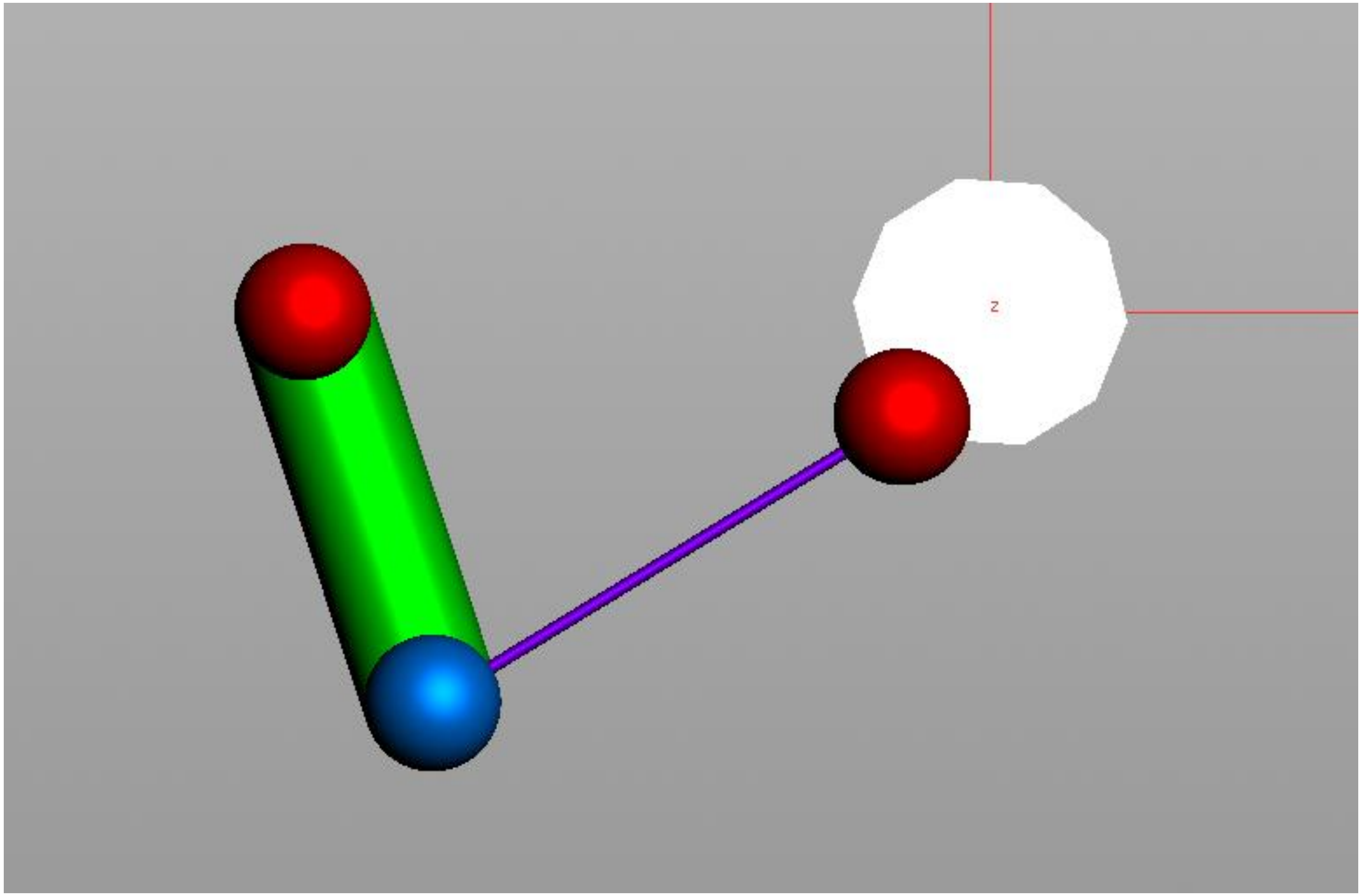
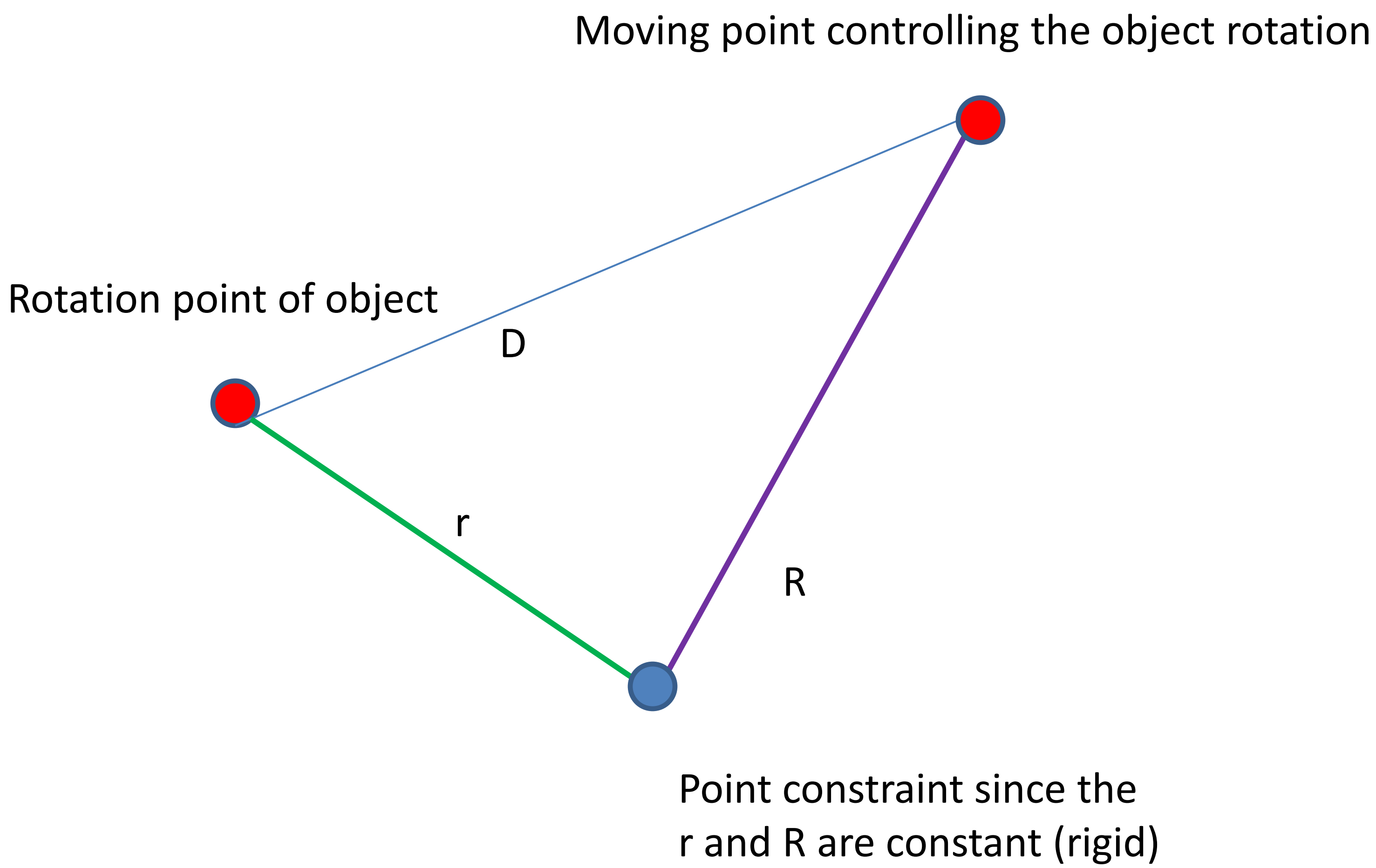
Dot product and pythagorean Theorem

by Deborah R. Fowler

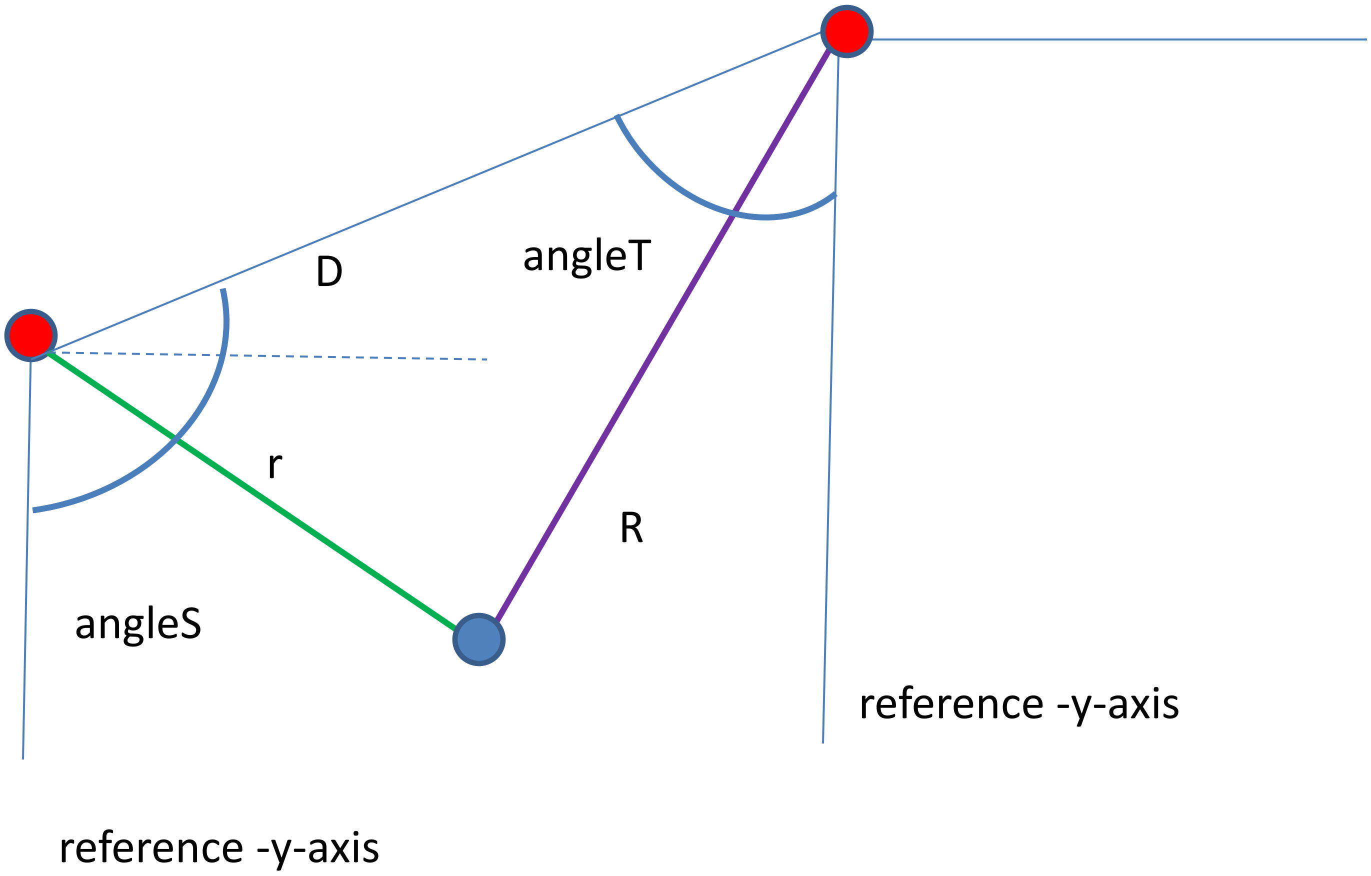
Similarly …



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



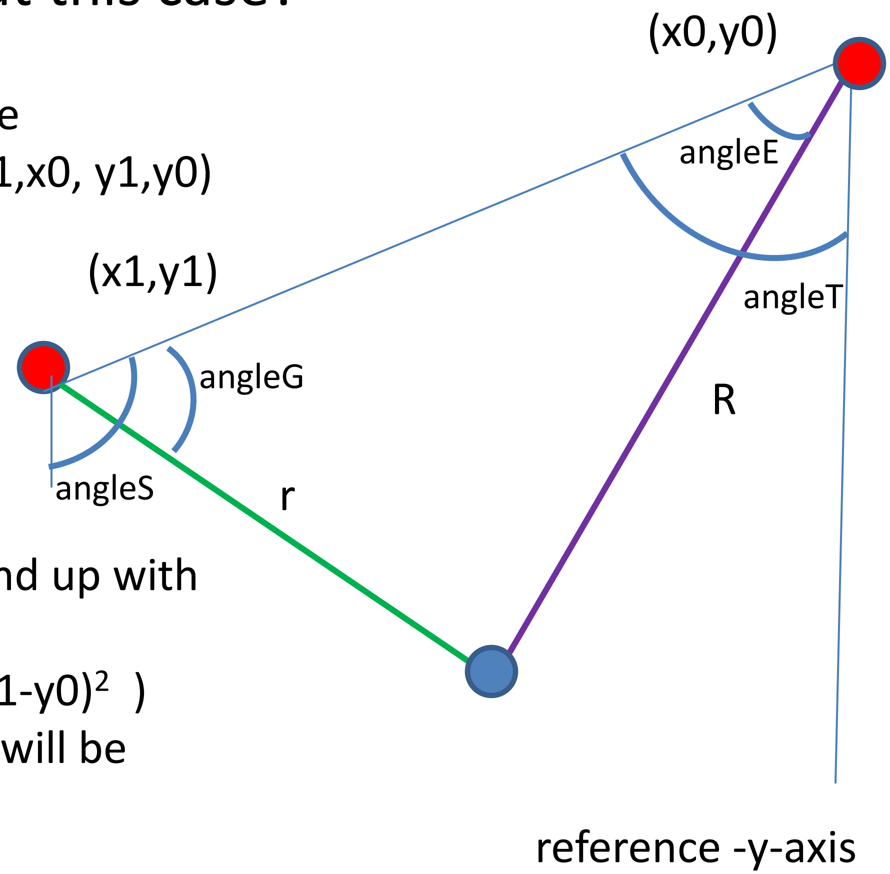
By definition, **dot product** of two vectors = cos (angle) * product of the length of the two vectors. We will use this property to calculate angleS and angleT, but before we do this, what else do we know?



What about this case?

Now our vectors are the reference negative y-axis, the other vecD is represented by (x1,x0, y1,y0) therefore the dot product is:

(0,-1) · (x1-x0, y1-y0)
0 * x1-x0 + -1 *(y1-y0)



Which is -y1+y0. If we normalize this we end up with
cos(angleT) = (-y1+y0) / (length of vecD)
angleT = acos((-y1+y0)/ sqrt($(x1-x0)^2 + (y1-y0)^2$)
Thus our rotation angle for the purple rod will be
270 – (angleT-angleE)

angleS = 180 – angleT (parallel lines)
Thus our rotation angle for the green rod will be
270 + 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  270 – (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
#
R = .4;
r = .3;
x1 = -.5;
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( ( -y1 + y0 )/D);
angleRot = angleT - angleE;
return  270 - 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
#
R = .4;
r = .3;
x1 = -.5;
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( ( -y1 + y0 )/D);
angleS = 180-angleT;

angleRot = angleS - angleG;
return 270 + angleRot;
}
```

in the sample file, dotPythagoreanInActionTopsyTurvy.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 negative y-axis is the reference axis

in the sample file, dotPythagoreanInActionTopsyTurvy.hipnc