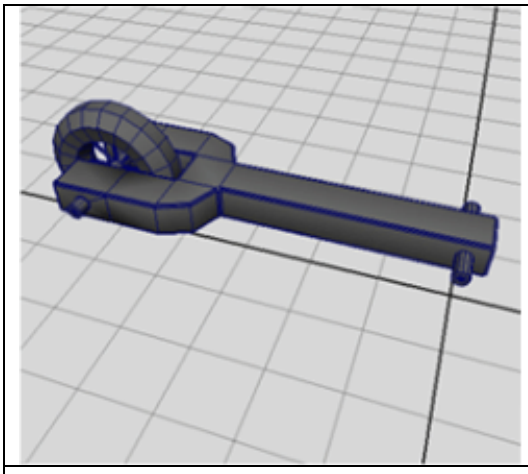
Working with Expressions & constrains

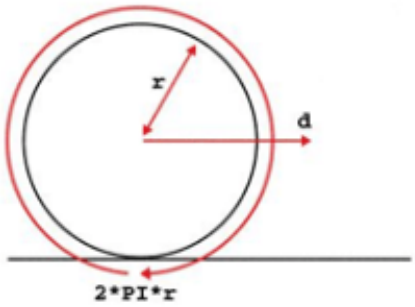
Expressions and constrains are extremely powerful and a riggers closest friend.
The idea behind this tutorial is to introduce expressions and show how to use constrains to help us in our rigging.
Expressions have many different applications. In this case we'll use them to make the wheel of a cart rotate automatically when we move the cart back and forth.
We'll expand the basics into making the cart tilt with some constrains and finish with a more complex example of a combination of expressions and constrains.



The cart
I built a little cart for the expressions exercise.

It's a simple cart with just one wheel and a handle that can be grabbed and moved around.

(see file cart_model.mb)



rotation/360 = d/(2*PI*r)
or
rotation = 360*d/(2*PI*r)

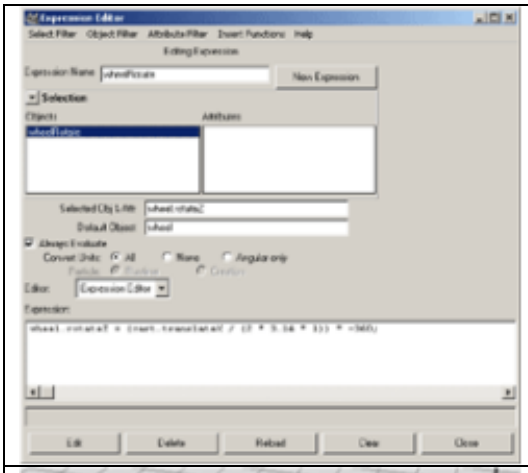
Turning the wheel
We'll write and expression that will turn the wheel when the cart is moved back and forth.

This expression will work only in one axis (in our case X)

To find out how much we have to rotate the wheel by, we have to use the formula:
rotation = 360 * distance / (2 * Pi * radius)

The radius we know is 1.

And the distance is the *translateX* of the cart.

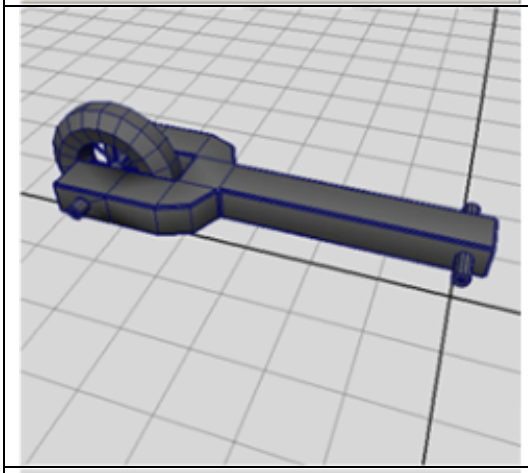


What it looks like
So, in the expression editor you can write the expression:

*wheel.rotateZ = (cart.translateX / (2 * 3.14 * 1)) * -360;*

We use the negative 360 because we need to rotate in negative Z.

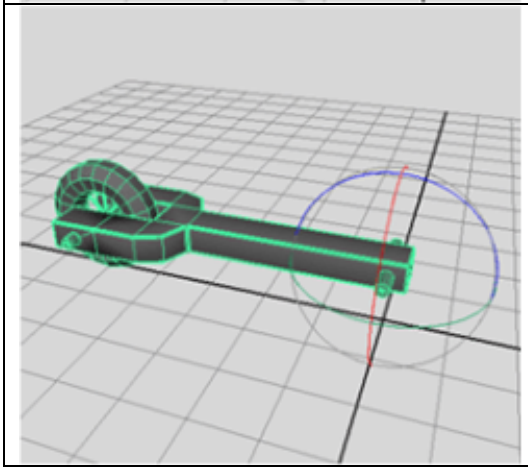
This is easy to find out. If your wheel is turning backwards, then you know that you have to turn it in the other direction, therefore, you times the whole equation by -1. Hence the -360.



That's it
That's all it takes to make the wheel turn on the ground while the cart is moved back and forth.

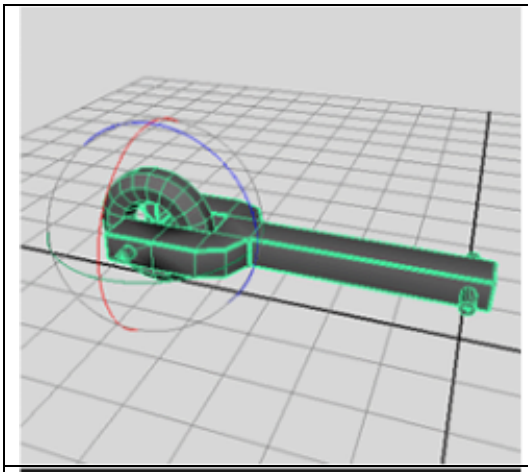
Easy.

(see file cart_exp.mb)



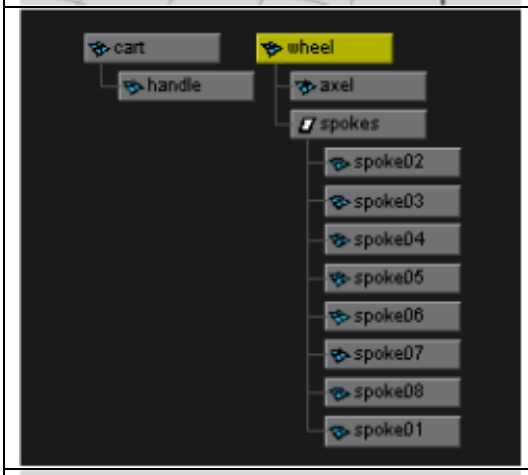
Moving on
Let's expand on our idea of the cart.
If we now would like to rotate the cart, you'll notice that we can't achieve this. The rotation axis for it is in the front, where the handle is, so it won't rotate around the axis of the wheel where you would expect the cart to rotate from.

To achieve this we'll have to re-build the cart a little and we'll use constrains to make sure that the cart rotates in the right place.



Changing things around
If we change the axis of rotation of the cart (by pressing ins) to where the axis of the wheel is, we get the rotate the cart the way we would like it to rotate, but that also means that you are rotating the wheel, and that's not what we want here.

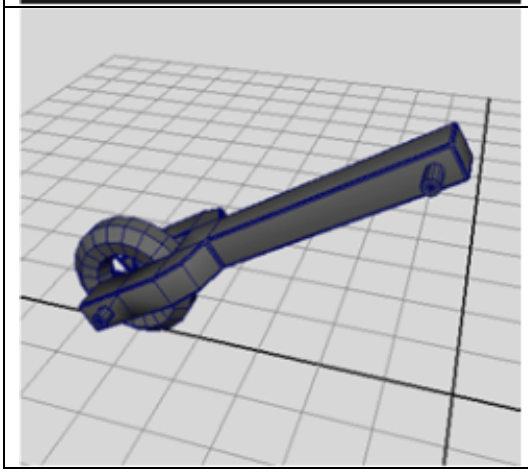
We would like to be able to rotate the cart, but the wheel to stay in the same place.



Breaking things apart
So, to be able to build the new modified cart, we'll have to break it.

I prefer to work with nulls and locators instead of the original geometry. This gives me a little bit more freedom to move things around, to see where the axis of rotation are and ends up being a much cleaner approach.

Un-parent the wheel from the cart. This will give us two different objects to work with.

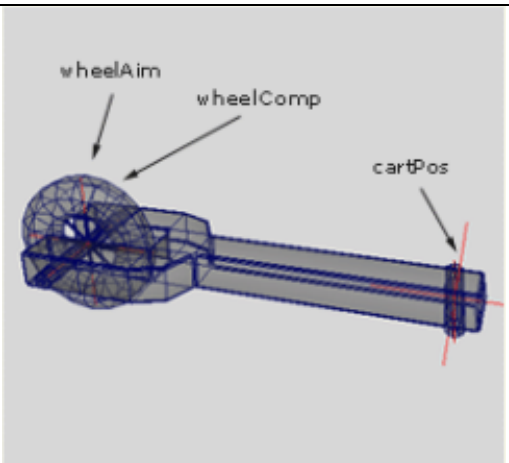


Changing the axis
Now we have two different object, the cart and the wheel. If we rotate the cart, the cart moves up and down and the wheel stays in place. But is we move it forward, the wheel stays behind.

If we add a point constrain from the wheel to the cart, we make sure that the wheel follows.

Perfect. In a lot of situations this is as far as we would like to go.

(see file cart_axis.mb)



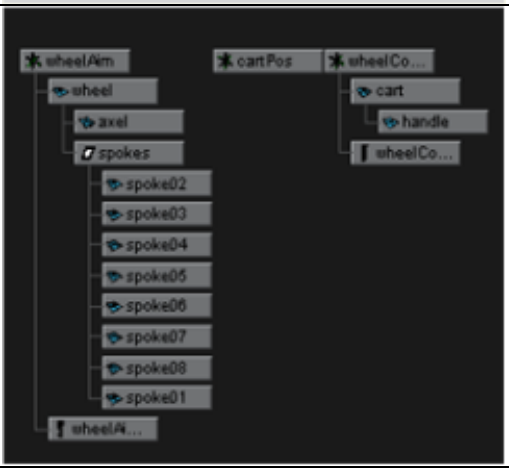
Building the rig
Let's move forward and try to build more on it.

Delete the point constrain that you just created.

Create three locators. Call them ***cartPos***, ***wheelComp*** and ***wheelAim***.

Position ***cartPos*** exactly where the handle of the cart is. This will be the object we'll grab to make the wheel move

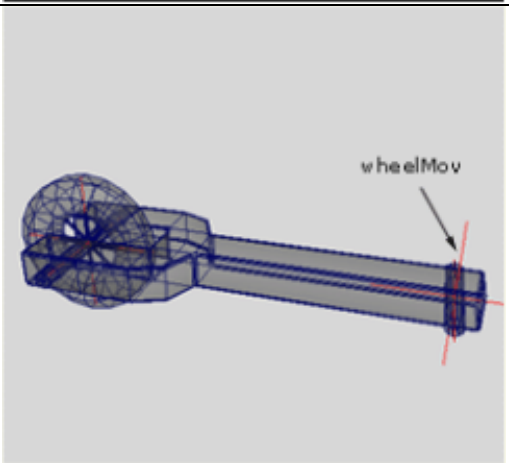
Position ***wheelAim*** and ***wheelComp*** in the axis of the wheel so that it shears it's rotation axis.



Constrains
Parent the wheel to the ***wheelAim*** and cart to ***wheelComp***

Point constrain ***wheelAim*** to ***wheelComp*** so that the wheel stays in the same position when the ***cartPos*** is moved

Also aim constrain the ***wheelComp*** to ***cartPos***. This will make the cart face the ***cartPos*** locator and hence follow it properly as we move it up and down.

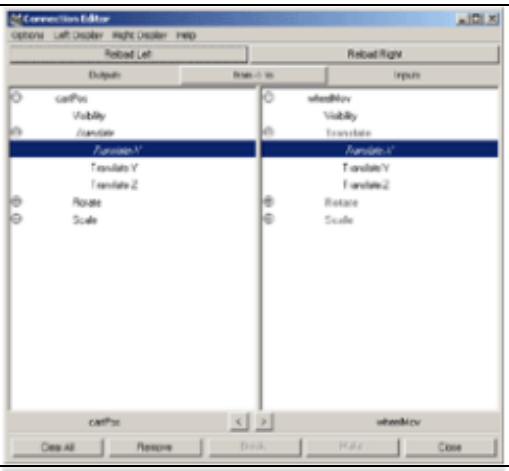
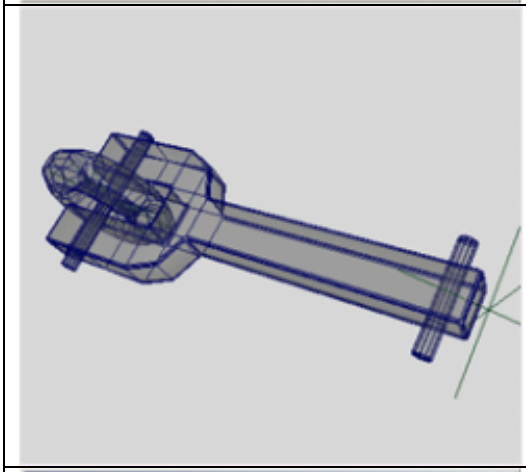
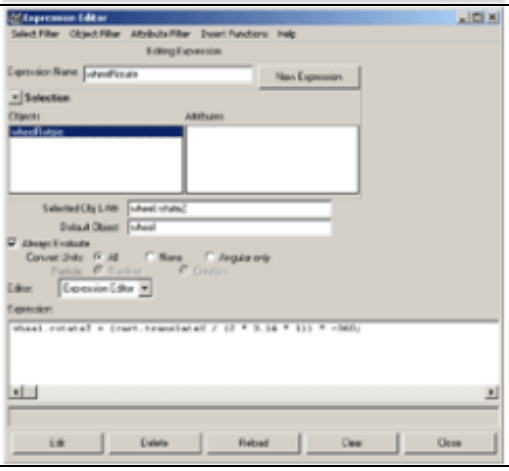


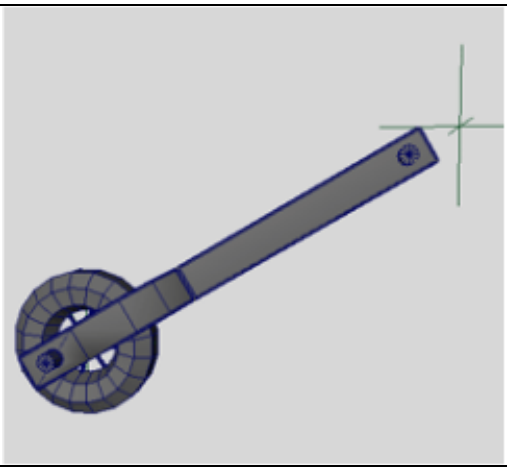
Moving forward
The cart now moves up and down when we move ***cartPos***.

So that the cart moves forward when the ***cartPos*** moves in X we need to connect the X translations.

Create an empty group and position it in the same place as ***cartPos***. Freeze the transformations.

Call it ***wheelMov***.

	**Moving forward** Parent ***wheelComp*** to ***wheelMov*** Connect in the connection editor ***wheelMov*** translation in X to ***cartPos*** translation in X. This connecting makes the wheel follow the cart on it's back and forth movement.
	**Fixing the expression** You'll notice that now the wheel follow forward when you translate ***cartPos*** in X and the cart rotates around the wheel when we move the cart in Y. But now we lost the rotation of the wheel. This is because the expression is connected to the ***cart*** but now we are moving ***cartPos*** instead. So we have to update our expression to ***cartPos*** instead of ***cart***.
	**Fixing the expression** If you notice that the wheel is turning backwards it's because we multiplied it by -1 before If we change that now from -360 to 360 that should fix it. *wheel.rotateZ = (cartPos.translateX / (2 * 3.14 * 1)) * 360;* *(see file cart_aim.mb)*

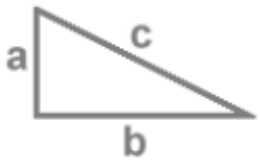


Moving on
Now we have the wheel turning and following when we move forward and the cart rotating on the proper axis.

But let's take it one step further.

You'll notice that when we move ***cartPos*** in Y that the handle of the cart and ***cartPos*** don't stay together. Basically, the wheel doesn't move forward to compensate for the upward movement.

Let's build something that will do this

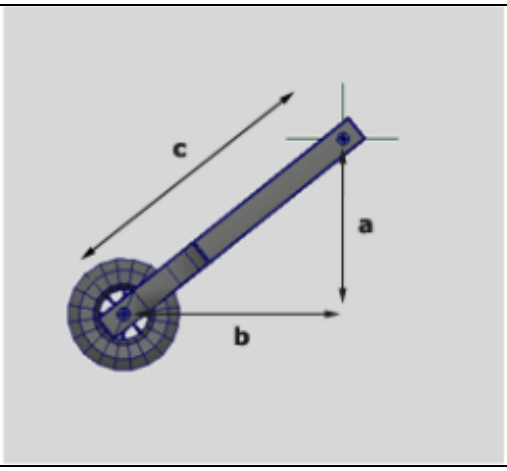


$$a^2 + b^2 = c^2$$

Pythagoras
To get to the next point, we'll have to use Pythagoras. His theorem states that.

If a triangle has sides of length (a,b,c), with sides (a,b) enclosing an angle of 90 degrees (right angle), then

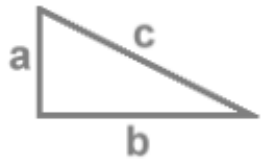
$$a^2+b^2=c^2$$



How does this help
Let's call ***a*** our translate Y for the ***cartPos***.

b is the translate X and this is our unknown, we would like to find out what ***b*** is so that we can move the cart that amount in X to compensate.

And ***c*** is the length of our cart, which we happen to know. So, ***c*** is fixed and in our case is 5.



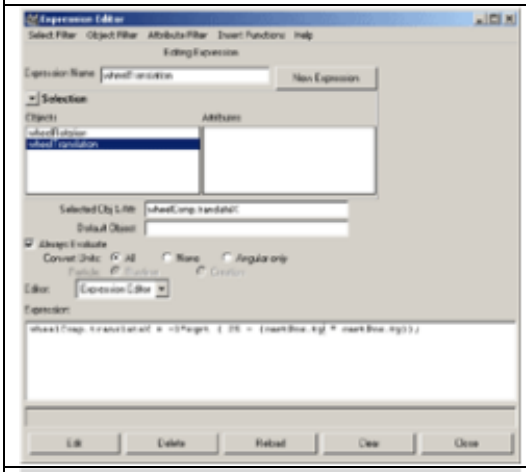
$$b = \sqrt[2]{c^2 - a^2}$$

Writing the expression
So, we need to find out ***b***.
If we do a little bit of simple math, we find out that:

$b=sqrt(c^2 - a^2)$

sqrt being the square root.

First of all, we need to freeze the transformations of ***cartPos*** so that everything is zeroed out.

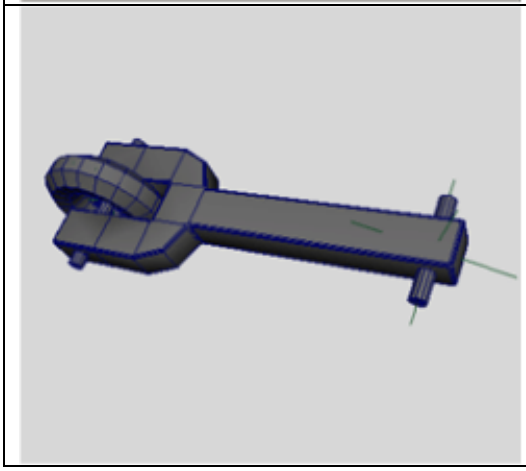


What it looks like

wheelComp.translateX = -1 * sqrt (25 - (cartPos.translateY * cartPos.translateY))

Once again we have to multiply the whole equation by -1 so that it aligns properly.

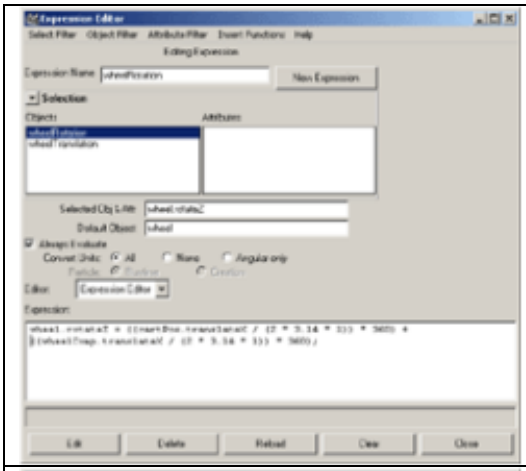
25 is 5 square and *cartPos.translateY* * *cartPos.translateY* is equal to *cartPos.translateY* square.



Almost there
Great, so now our wheel is following when we move the ***cartPos*** up and down.

We are very close. You'll notice now that two things happen.

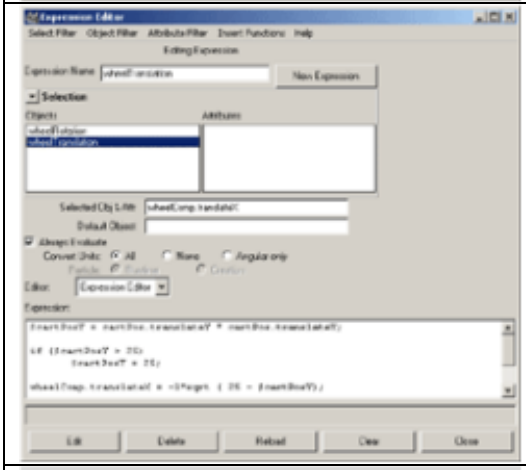
1) the wheel doesn't rotate when it moves up and down
2) When we move ***cartPos*** beyond 5 or -5 in Y we get an error in the Command Feedback and the wheel doesn't really align properly. This is because we are trying to get the square root of a negative number.



Fixing the wheel
Let's update the wheel expression to compensate for the changes we've done.

```
wheel.rotateZ = ((cartPos.translateX / (2 * 3.14 * 1)) * 360)
+ ((wheelComp.translateX / (2 * 3.14 * 1)) * 360);
```

What we've done is add rotation when the ***wheelComp*** translates in X.

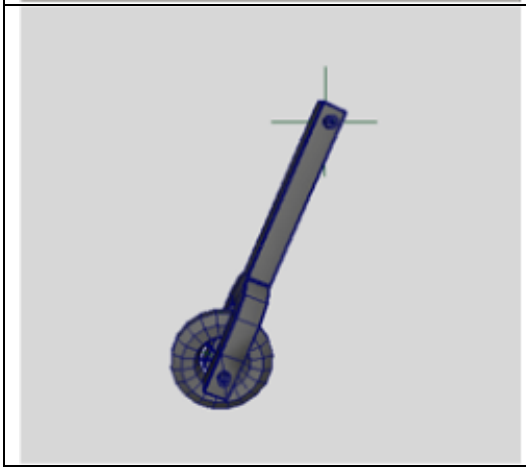


Fixing the error
Let's update the wheel expression to compensate for the changes we've done.

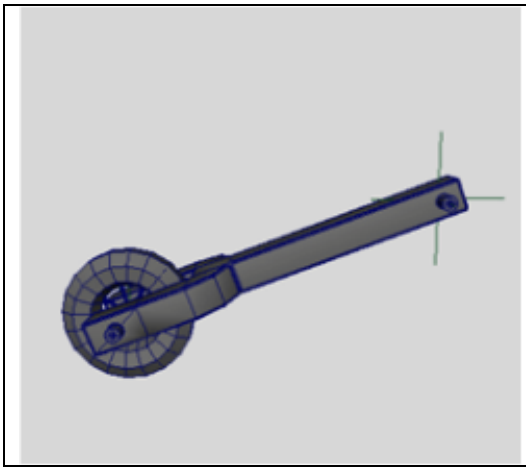
```
$cartPosY = cartPos.translateY * cartPos.translateY;

if ($cartPosY > 25)
        $cartPosY = 25;

wheelComp.translateX = -1*sqrt ( 25 - $cartPosY);
```



What we are doing
If we try to get the square root of a negative number we get an error, so what we are doing here is calculating the square of the *cartPos.translateY* and if it's greater than 25 (which is the length of our cart squared), then we force it back to 25. This way we get 25-25 which is 0. The square root of 0 is 0. If we multiply it by -1 we still get 0, hence the cart doesn't move in X.



Done
That's it.

Now we have the cart following our locator ***cartPos*** wherever it goes and the wheel rotating accordingly along the ground. (as long as you are moving along the X axis)

(See file cart_done.mb)