

Strata CX 5 Lua Scripting

Creating an Aim-Constraint

October 3, 2006
Provided by Jon Bradley

Table of contents

Scripting an Aim Constraint

Purpose.....	3
Setting Up Your Scene	3
Creating The Target Script.....	4
Creating the Aim Script	7
Recap.....	10
Discussion	10

Scripting an Aim-Constraint

PURPOSE

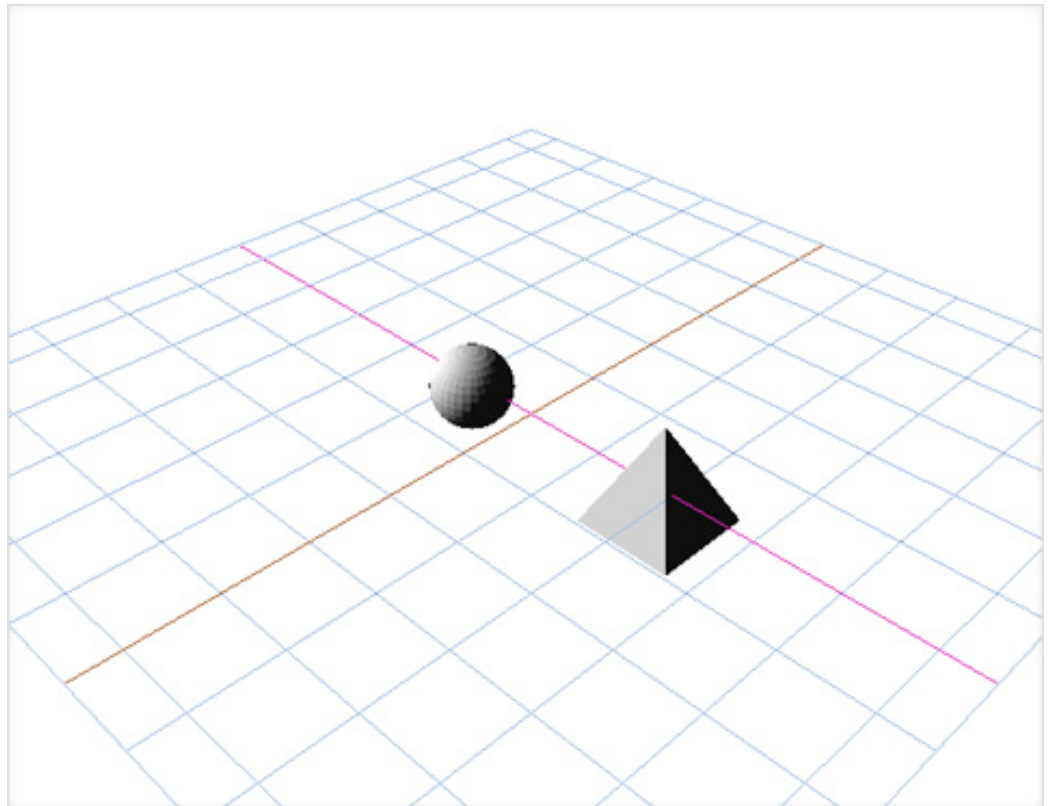
This tutorial is to show how to create an aim-constraint — where one object aims at another object at all times — in Strata CX 5 and greater using standard Lua scripting in the shipping version of the product as of the date of this tutorial.

This tutorial does not assume familiarity with Lua, but an understanding of Strata CX 5 is necessary to complete the tutorial.

SETTING UP YOUR SCENE

No specific setup is required. For the purposes of this tutorial, we will use an animated sphere as the target and a pyramid as the pointing (aim) object. You can use any object you wish, but a few details we'll point out here will need a piece of geometry that can easily show rotation and a pointing *direction*. A pyramid is a perfect aim source for this.

Here's an example screen shot of an initial scene setup.



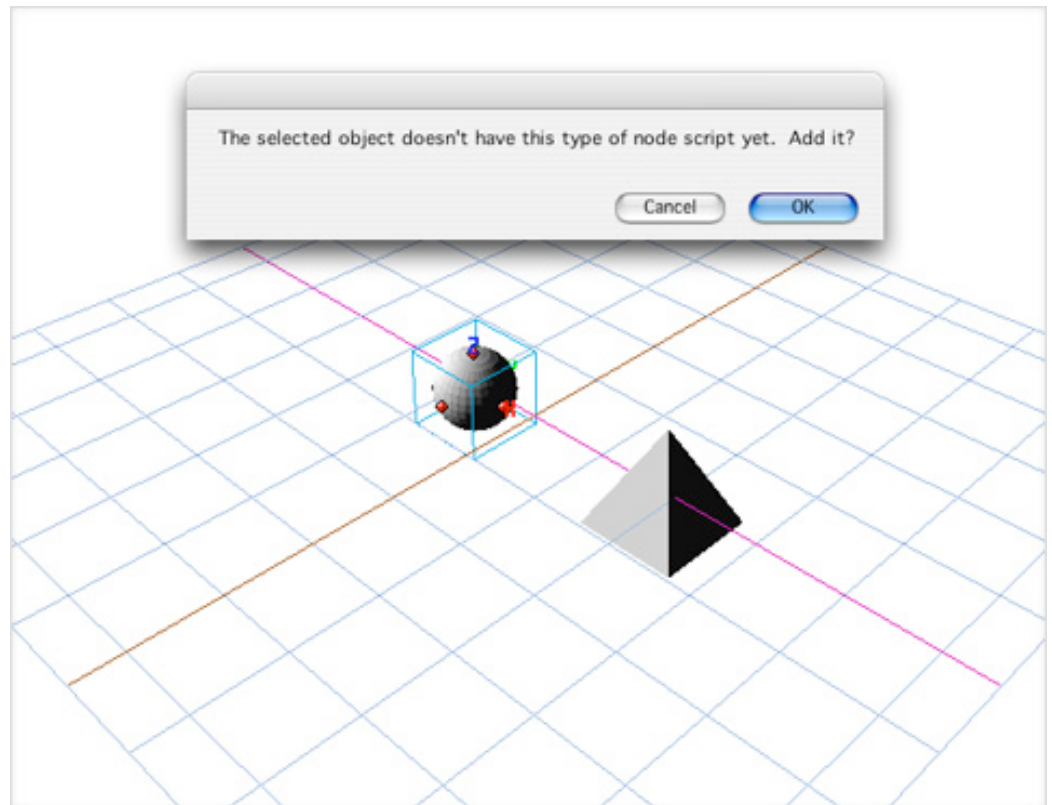
For this tutorial, we will be using the concept of *Node Scripts*, which are scripts added to a node of an object. The node may be a position, scale, rotation, or any other type of node that allows scripting. You can determine whether a node of an object may be scripted by looking for an “=” icon in the Project Window, next to the node in question.

We will be using a nice shortcut provided in the Scripting Menu of CX 5 that will allow us to add a new node script to an object relatively quick and easy.

CREATING THE TARGET SCRIPT

The first required part to creating an aim-constraint is to share the position of our target so that any object that wants to aim at it knows where it is in 3d space.

Select the sphere and open the menu *Scripting > Instance Nodes > Edit Position Script*. This will most likely pop up a warning dialog, as seen below:



Select OK when the dialog appears. If you already had a node script on the object you selected, you would be taken directly to the editor for the script.

ADDITIONAL INFORMATION ON THE SCRIPT EDITOR

There are two areas to enter scripts in the Script Editor, *Initialization* and *Script Source*.

The Initialization script is a one-time script that will run when you add the script to the object (by confirming the script editing session with the OK button). Note that if you make changes to this script, it will run again. This allows you to create scripts that may include helper functions or any additional information needed during the *run-time*, or Script Source in the lower edit box.

The Script Source may also be called the *run-time* script, as it is executed constantly during scene playback and at each frame change during a rendering.

For us to share the position of our target object, we need to create a global space to share the function that returns the position of the object at a certain point in time.

Type (or copy/paste) in the following in the Initialization:

```
s3d.whiteboard = s3d.whiteboard or {}  
s3d.whiteboard.aimConstraint = s3d.whiteboard.aimConstraint or {}  
local targetA = "sphere"  
s3d.whiteboard.aimConstraint.AimAtObject = targetA
```

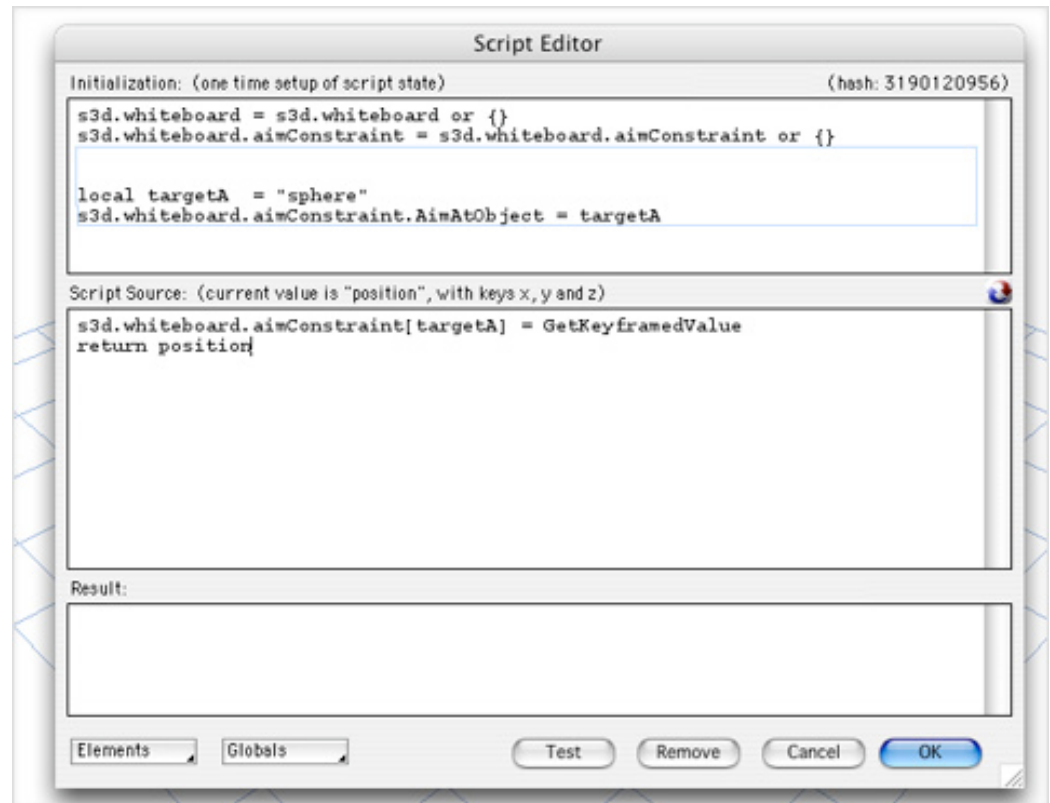
The above script creates an object, `s3d.whiteboard.aimConstraint.AimAtObject`, in the global Lua table. We use a local variable, `targetA`, so that the Script Source code will be able to reference it when we share a function to return the position of our target at any point in time.

Type (or copy/paste) in the following in the Script Source:

```
s3d.whiteboard.aimConstraint[targetA] = GetKeyframedValue  
return position
```

The above function finalizes our target by placing the function `GetKeyframedValue` in the Lua table we created so that our aiming object may call it at a later time. The `GetKeyframedValue` function will return whatever value is being edited in a node script (position, rotation, scale, etc.) if it is passed `time` as a variable, as in `GetKeyframedValue(time)`

Our final script for our target should look similar to the screenshot below.



CREATING THE AIM SCRIPT

To have our aiming object (the pyramid) point at our sphere, we need to add two script nodes to it: a position script and a rotation script. The position script will allow the pyramid to use its own position in the rotation calculation. *Aside: If you were wondering, yes, that access definitely needs to get simplified in the future.*

The rotation script is where a small amount of math will be used to calculate our rotation (a Quaternion) that will be used to point at the target.

Following the same procedure above for the sphere target (**Select the pyramid** and open the menu *Scripting > Instance Nodes > Edit Position Script.*), add a position script to our pyramid.

Type (or copy/paste) in the following in the Script Source:

```
s3d.whiteboard.aimConstraint.aimPosition1 = GetKeyframedValue
return position
```

We do not require any initialization for the pyramid, because we can just create the shared variable directly to our global table we created in earlier steps (`s3d.whiteboard.aimConstraint`). Note that the script for the target could also be simplified, but we are using the Initialization area for the target to create our initial tables.

Next, we'll add the final portion to our script, the rotation node: **Select the pyramid** and open the menu *Scripting > Instance Nodes > Edit Rotation Script.*

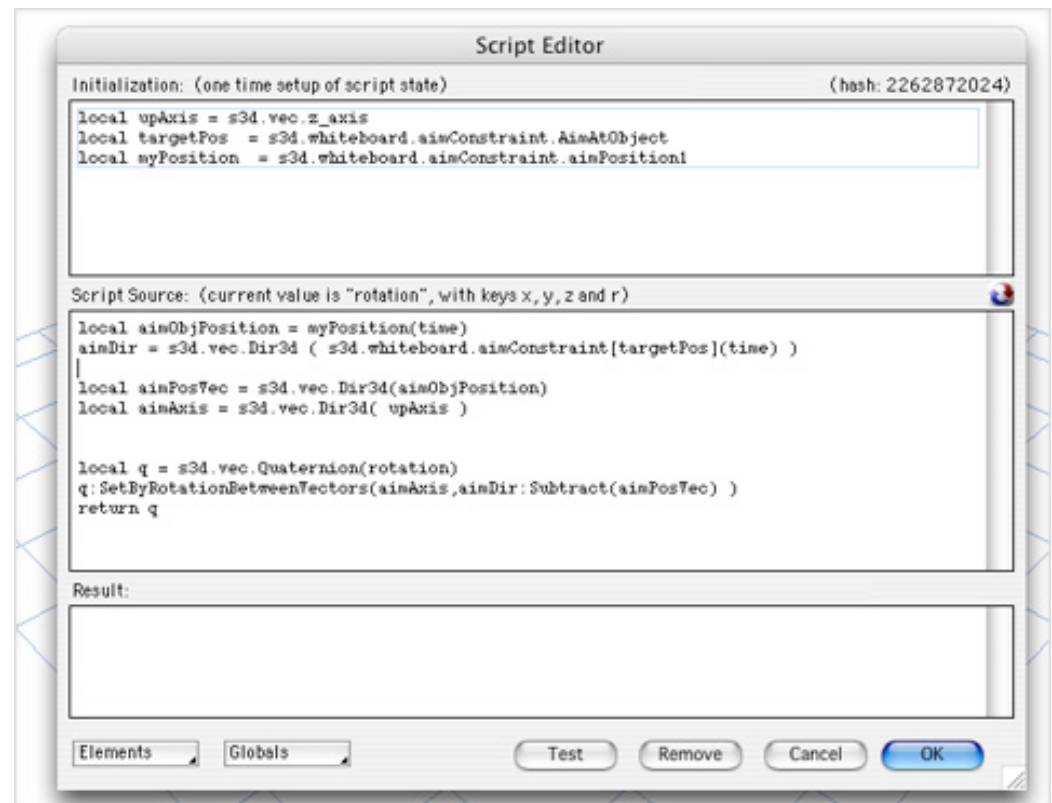
Type (or copy/paste) in the following in the Initialization:

```
local upAxis = s3d.vec.z_axis
local targetPos = s3d.whiteboard.aimConstraint.AimAtObject
local myPosition = s3d.whiteboard.aimConstraint.aimPosition1
```

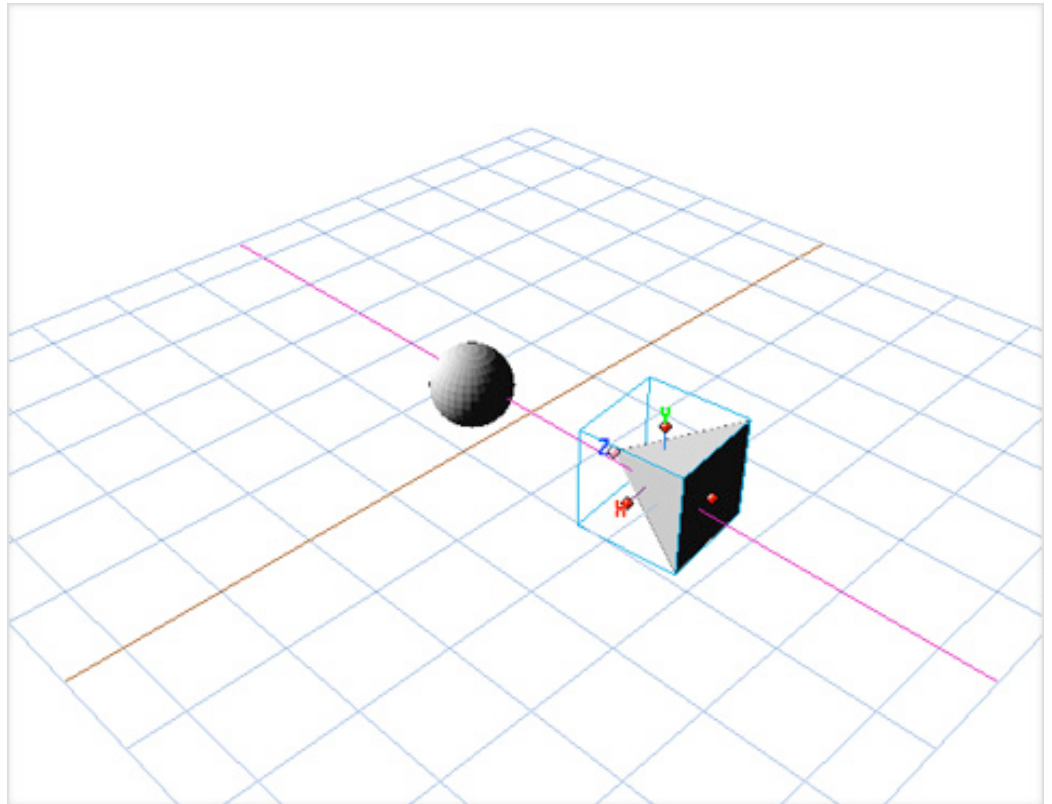
We use the `upAxis` to help us calculate a pointing direction for the object (we will not get in more detail with this value right now). The target position we grab from the global table we created earlier so we can use it local to our pointing object, specifically in the Script Source (run-time). The same can be said for the shared position of our pyramid, which we created just a bit ago.

Type (or copy/paste) in the following in the Script Source (final script in image below):

```
local aimObjPosition = myPosition(time)
aimDir = s3d.vec.Dir3d (s3d.whiteboard.aimConstraint[targetPos](time))
local aimPosVec = s3d.vec.Dir3d(aimObjPosition)
local aimAxis = s3d.vec.Dir3d( upAxis )
local q = s3d.vec.Quaternion(rotation)
q:SetByRotationBetweenVectors(aimAxis,aimDir:Subtract(aimPosVec))
return q
```



Select OK. The pyramid should now be pointing at your sphere object. See the figure below for what the scene included with this tutorial PDF looks like once all of the scripts were added and run.



The final rotation script in the pyramid performs the following steps:

- Executes the function `GetKeyframedValue` stored earlier for our object we are aiming and stores the position (a `s3d.vec.Point3d` value)
- Calculates a directional vector using the initial values from the position of our target object (`aimConstraint[targetPos](time)`)
- Creates a temporary Quaternion using the current rotation value that is available in the Script Source, and calculates the aiming vector by using the up axis and the difference between the target and the aiming object current facing directions.

Finally, feel free to now animate the sphere and watch the pyramid aim at it during interactive playback. Also, you can animate the pyramid as well, having it constantly aim at the sphere no matter where it travels in 3d space.

RECAP

The scripts shown in this tutorial are a basic foundation for an aim-constraint setup in Strata CX 5. This overview will hopefully give you some ideas how you can create your own constraints (similar to bones, rotation constraints, pivots, etc.).

The script shown here does have its downfalls: the pyramid used will tend to spin around its original up axis as it tries to keep itself oriented. This is normal for the script we used since we did not implement any method to lock the up direction while the script is running. We just used a generic up axis in the z-direction to assist in the pointing.

This script could be extended to provide other interesting capabilities, such as aiming textured cards in a scene to a camera (such as trees or textured people cards in an architectural rendering). Although the end result is fairly simple, I hope that this overview gives you some more insight into what you can accomplish with some scripting in CX 5.

DISCUSSION

If you haven't seen it yet, **StrataCafe.com** has a Lua Scripting forum devoted to scripting in Strata CX 5. If you have any questions on this script, additions to it or anything else you want to discuss, please head on over and join the forums.

Have fun scripting!