Ai Example

Moving Screens

The background of a stage is formed of screen segments, each hung from Artnet controlled winches. Ai calculates position and rotation of each segment, and maps the contents accordingly.

by:	Sebastian Beutel, January 2016
published:	here
tested in version:	Ai v8
download:	Stagepatch , Models The models are distinct as for their UV mapping - make sure to load the correct model into each screen.

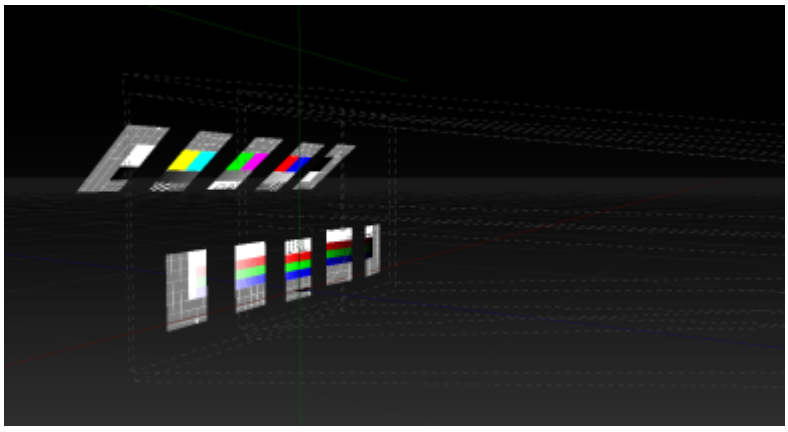
Hint: click the images to show them larger.

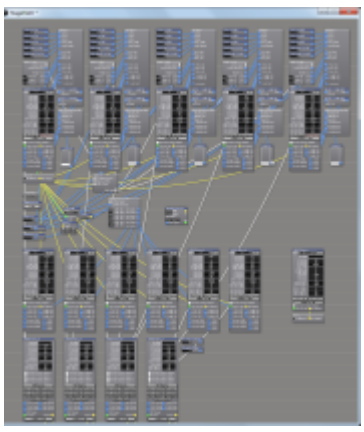
moving, winch, quaternion, uv-map

Background

This project was done for a friend of mine who is the lighting designer for one of Germany's most well-known comedians. He designed the stage elements and wanted to use Ai to map on the moving segments. This project was used on tour for more than a year.

The background of the setup is formed from a number of segments, each a flat surface of approx. 1 x 2 meters. Five such segments are hung from the rig, each with 3 Artnet-controlled winches. Per segment there is one winch for the top/left corner, one for the top/right corner, and one for the bottom-center point, this winch being suspended more upstage. With the winches in such a setup it is possible to hoist the segment, and to rotate and/or tilt it (within limits). The winches used in this setup feed their current position back via Artnet - and these data are used by Ai to calculate the current position and rotation of each segment. Likewise, more such segments were standing on the ground, on Artnet-controlled rotators, which also fed their current position back. This way it was possible to map contents onto the whole surface.





Here, Ai was not only used as visualiser, but was the main mapping machine. The show was controlled from a GrandMA2 command wing, and most of the parts were timecoded to be in sync with the music.

Some of the special aspects of this project:

- it was necessary to mimic the real behaviour of the segments, in order to reproduce it from the position data
- something what was completely new to me: we ran into a gimbal lock, and Mr. Dave Green was kind enough to explain this and provide a solution - see EulerToQuaternion
- the handles of the projectors and of most of the screen fixtures were hidden from the stage construction page, in order to gain overview
- it is not possible to do a proper softedge on surfaces which can dynamically tilt. Hence it was decided to go for a hardedge, and only some segments were displayed per projector
- as the entire background was always mapped as one big canvas, only one screen was exposed to the Ai GUI. The part each segment was to show was set in the UV map - hence, each segment needed to have its designated model.

Used Modules and Patches

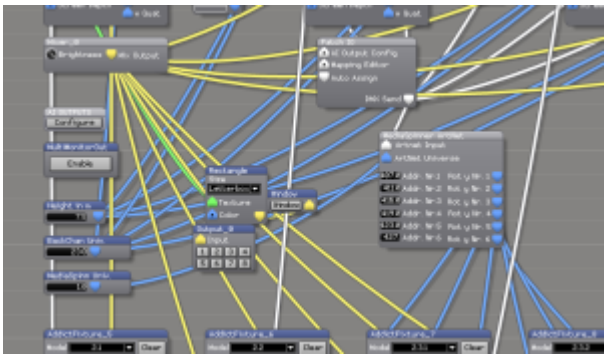
- Constant
- Rectangle
- Window
- video out
- Monitor
- Euler To Quaternion
- Formula
- ArtNet Input Large
- Patch IO
- Notepad
- Vector Math

Stage Patch

Again, the whole stage patch can be devided in some sections:

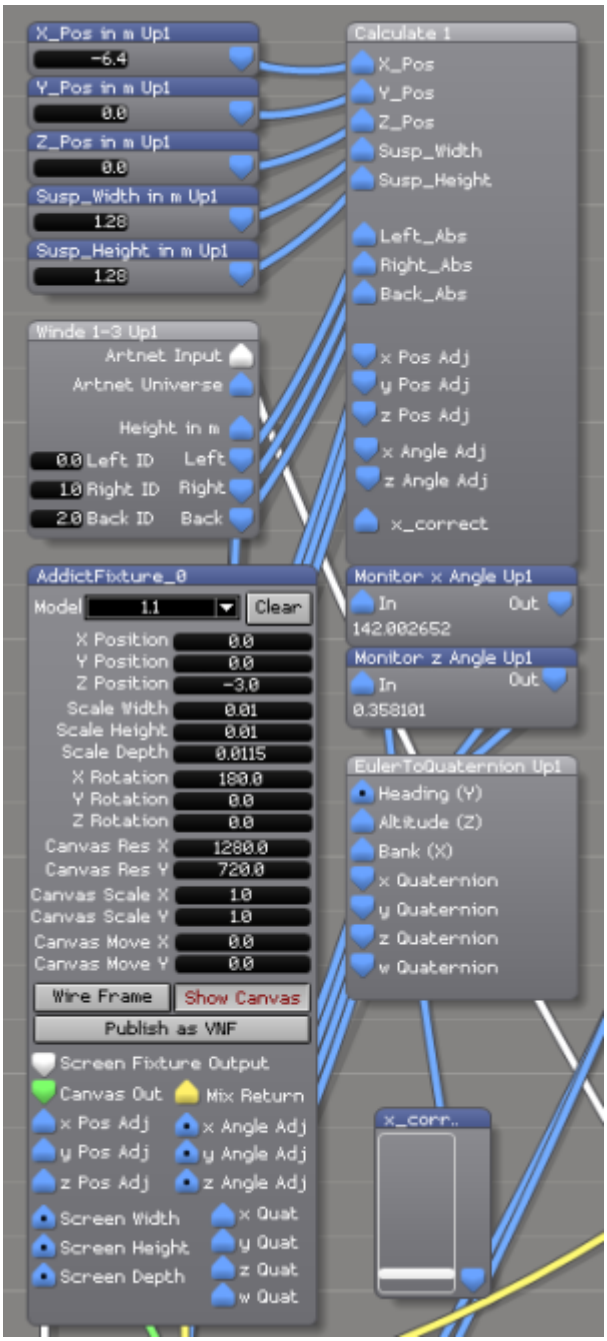
- in the middle there are the main inputs and outputs
- the upper part holds the upper (hung) segments - the screens, inputs, and calculations
- the lower part holds the segments which were standing on the ground
- at the bottom there are the projector definitions

Middle: the main inputs and outputs



- there are a few Constant modules to make it easy to set some parameters globally:
  - `Height in m` sets the general height of where the winches are hung
  - `BackChan Univ.` sets the ArtNet universe for the data for the background segments' winches
  - `MediaSpinn Univ.` sets the ArtNet universe for the rotators on the ground
- the little cluster of a Rectangle, a Window and a video out (`Output_0`) gets its signal from the first screen fixture (which is the only one shown in the GUI) - this allows a quick preview, in a window or on a screen
- the `AI OUTPUTS` patch and the `Multi MonitorOut` button are simply moved here when moving modules and patches around - they belong to any proper stage patch/project
- `Mixer_0` is the only mixer in this patch - it stems from the beginning of this project when only one screen was in the patch. Note that as there was only one mixer needed (all screen elements get the same contents and apply their UV map), all other mixers were simply removed, and all screen fixtures get their video input from this one mixer
- `Patch IO` is not the module but the main IO which is there in every project, to allow connectivity to your network. Here, the `DMX Send` port needs to be connected to various other patches in order for them to receive the ArtNet data
- the `MediaSpinner ArtNet` subpatch belongs (technically) to the ground rotators and is explained further down

Upper part: the hung segments



For each of the 5 hung segments there are some patches and modules in order to calculate positions and rotation data, and to display the screen segment:

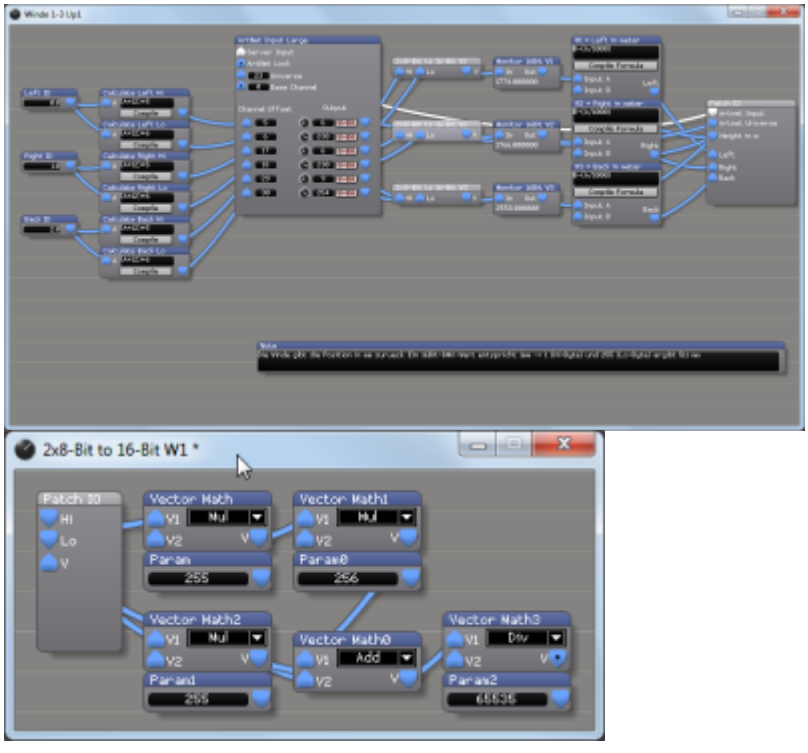
- top-left, there are some constants to define the general position: `X_Pos`, `Y_Pos` and `Z_Pos` set where the segment generally is (its origin), and `Susp_Width` and `Susp_Height` let us fine-tune how the segments are rigged (the smaller the suspension distance, the more tilt can be acchieved with a little winch travel)
- `Winde 1-3` is a subpatch. It takes the global `Artnet Input`, `Artnet Universe` and `Height`. Also, the IDs of the winches are exposed as inputs and need to be set to match which winch is located where. The patch then takes the Artnet data and calculates the height of each of the three points (left, right, back). These results are then send to the next subpatch `Calculate`. For an explanation of the calculations see below.
- `Calculate` is the subpatch where the magic happens: from the height data for each point the absolute position and rotation of this segment is calculated. Inputs are the constants defined and the positions from the previous subpatch - outputs are position and rotation (angle) data.

For an explanation of the calculations see below.

- two Monitor modules were inserted here only to monitor the computed angels
- Euler To Quaternion is another subpatch which is explained below. It converts the computed rotation data to another dimension (Quaternions).
- `x_corr` is a fader to allow for minor corrections of the x position - its value is also sent into the Calculate subpatch
- finally, bottom-left is the screen fixture for this very segment:
  - only the first one is left with its original name `AddictFixture_0`, and all others were renamed to some other name. This is how Ai works: only fixtures with names in ascending order are displayed with handles in the GUI. Renaming the other screen fixtures removes their handles, but lets them still show up, in the GUI
  - make sure to load the correct model into each screen fixture (drag and drop the 3ds file onto the fixture)
  - `Canvas Res X` and `Canvas Res Y` need to be set to the values the contents is produced in
  - `Screen Fixture Output` is sent to the projectors at the bottom of the stage patch
  - the green `Canvas Out` is connected only for the very first screen (the one with the mixer) and is patched to our little monitor window (see above)
  - `Mix Return` gets the main video mix from our only mixer (see above)
  - `x/y/z Pos Adj` are connected to our `Calculate` subpatch where they get the positions from
  - `x/y/z/w Quat` get the special rotation data from the `EulerToQuaternion` subpatch

Winde 1-3: getting the absolute height data

As described above the subpatch `Winde 1-3` takes the Artnet dat and returns the absolut heights for the three points. Double-click the module to reveal its contents:



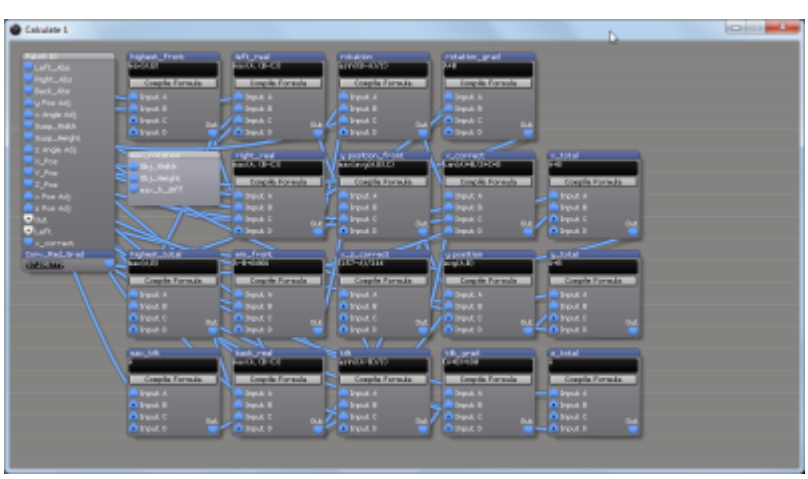
This should be easy to understand - simply read it left-to-right:

- the (exposed to the parent patch) constants take the Id per winch, and based on this, the following Formula modules compute the two channel numbers per winch
- the ArtNet Input Large module (with the `Server Input` and `Universe` ports patched to the Patch IO) return the Artnet values for the 6 channels in question (here it was decided to get the values normalized, i.e. 0..1)
- the `2×8-Bit to 16-Bit` subpatch uses a little Vector Math to combine the high and the low channel into one value
- of course, for debugging, some Monitor modules display the results
- the Formula modules convert the results to meters (as the winches have a resolution of 1/millimeter), and subtract this from the global height (again from the patch io)
- the result is then returned via the patch io
- the Notepad simply gives some explanations: '*The winch returns its position in mm. One Artnet digit equals one mm travel. E.g. 1 (hi byte) + 255 (lo byte) gives a travel of 511 mm.*'

Calculate

`Calculate` is the main subpatch where from the three points' height the position and rotation is computed.

Note that this is not exact - honestly I had wished to have someone with a degree in physics to help me with this. However it was close enough to reality to be used on a tour for more than a year.



Let's try to explain the logic:

- `highest_front`: the highest front height (from *Left_Abs* and *Right_Abs*)
- `max_rotation`: the max. height difference based on suspension width/height
- `left_real`, `right_real`: either *Left_Abs* (*Right_Abs*) if this is higher, or (if this is the lower point) the other point's height minus *max_rotation*
- `highest_total`: the greatest height - either from *highest_front*, or *Back_Abs*
- `max_tilt`: right now this directly passes *Susp_Height* but is already a formula module for further tweaking
- `back_real`: the greatest height from either *Back_Abs* (if the back point is the highest) or the highest point from L, R, B, minus the suspension height
- `min_front`: (this is in case B would be the highest point - it was agreed that this would never happen by controlling the travel properly, to prevent the pane from flipping): *back_real* minus *Susp_Height* plus 0.001

- `rotation`: the 'roll', computed from *left_real*, *right_real* and *Susp_Width* – note that this result is in radiants
- `rotation_grad`: converts the rotation (roll) value to degrees (by multiplying with 180/PI)
- `y_position_front`: either the mid between *left_real* and *right_real*, or *min_front* (if L and R are slack)
- `tilt`: the tilt angle is computed (in radiants) from diff between *back_real*, *y_position_front*, and *Susp_Height*
- `tilt_grad`: by multiplying with 180/PI tilt is converted to degrees. +90 accomodates for rotated models. The result is *x Angle Adjust*
- `x_z_correct`: the amount of correction for the x value, based on the tilt angle
- `x_correct` is the amount of correction to the x value (how far the model is shifted horizontally). This is computed from *rotation* (roll), *Susp_Height*, *x_z_correct* and *x_correct* which allows to be set from the parent patch
- `x_total`: *X_Pos* and *x_correct* give the total x value (*x Pos Adj*) - this is the left-right-position based on the computed rotation angles
- `y position`: the amount of correction for the y value, based on the computed heights of the front (*y position front*) and back (*back real*)
- `y_total`: *Y_Pos* and *y position* give the total y value (*y Pos Adj*) - this is the top-bottom-position based on the computed rotation angles
- `z_total`: it was planned to add some logic to also correct the z parameter (front-back) - however, here *Z_Pos* is only passed to *z Pos Adj*

EulerToQuaternion

When creating this project I ran into a strange problem: x, y, and z rotation are no independent from each other. I described it like this:

> in AI, the x and y rotation of a screen fixture refer to the world's x and y axis. However, z rotation refers to the screen's z axis. This leads to the situation – with x_rot = -90 – that y_rot and z_rot do exactly the same, but the screen cannot be tilted sideways (what z_rotation does).

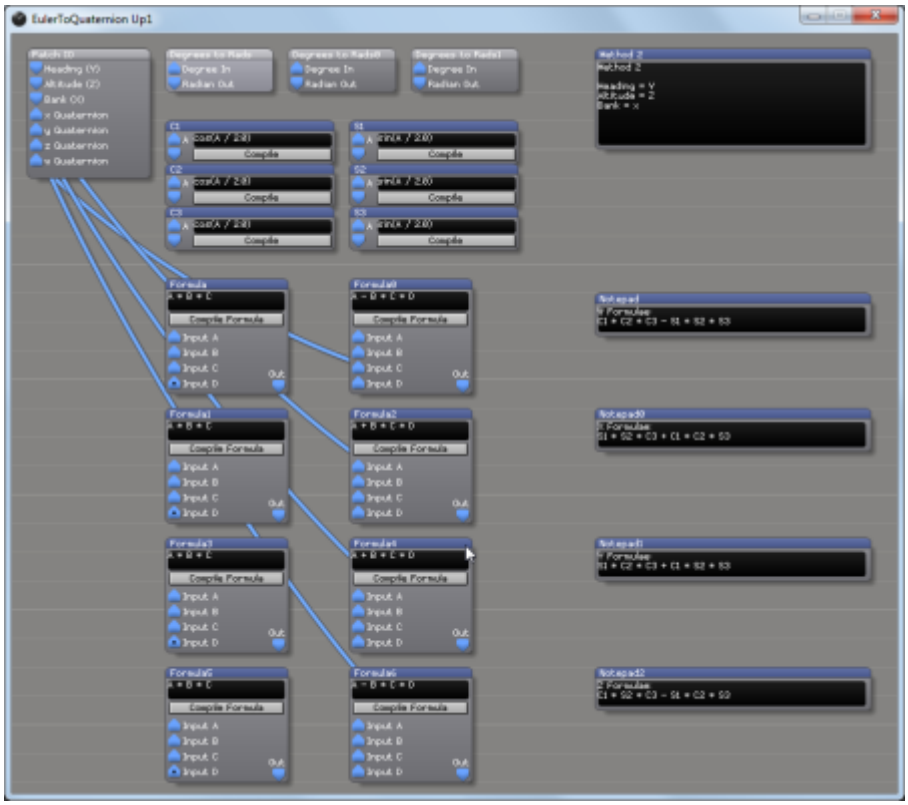
Dave Green was kind enough to give a thorough explanation of this:

> You have discovered one of the draw backs of the Euler angle system we use in to rotate screens in Ai. Gimbal Lock. There is a short video explaining the problem here: https://www.youtube.com/watch?v=zc8b2Jo7mno. The advantage to using the Euler system for angles is that everybody understands it. But the trade off is that it has limitations in the form of gimbal lock.
>
> So you are probably thinking what can I do to resolve this problem? Well the simplest option is to have a second version of your model which is 'pre-rotated' by 90' on the x axis. That will allow your z rotation to function as you expect. That might not be the solution you need in this case, but it is a common solution to the problem.
>
> Another option is to use the alternate skin for the Fixture in the stage patch and then connect to the x,y,z,w Quat ports. I believe these allow you to rotate the fixture using quaternion rotation. Only problem is I just tried this and I have no idea what data you need to feed into it. I know this solves this problem though if you can work out what to feed into it. Ciaran may be able to elaborate here as he added these ports. There is a short video explaining quaternion rotation here: https://www.youtube.com/watch?v=SCbpxiCN0U0 you should be able to take some math / trig

> modules in salvation and replicate the conversion detailed in this video if you are feeling brave ;)

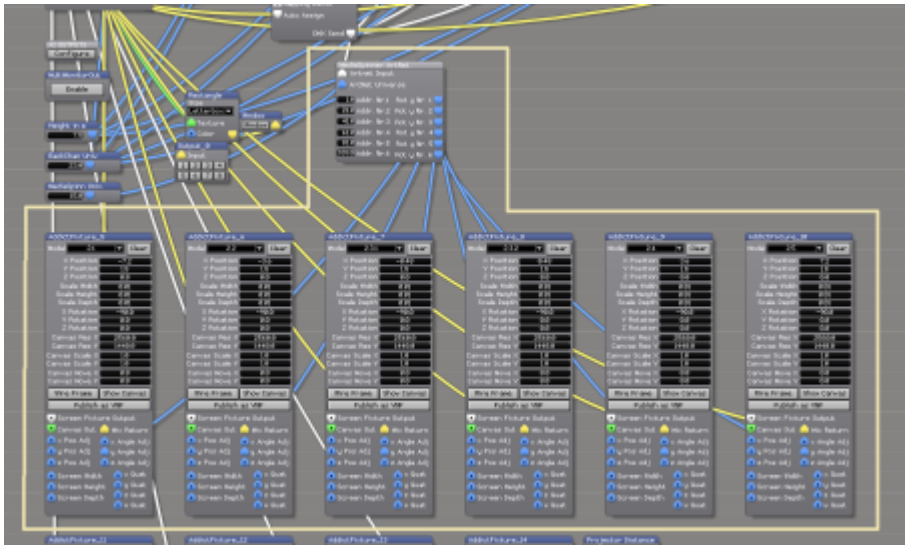
And while I was still struggling to find my way through all this Ciaran was so kind to create a patch which is now included as system patch:

> As Dave has said you encounter this problem based on the way Euler angles are defined in Euclidean Space. To get around this we can either use a 4×4 rotation matrix or quaternions. Without getting into the maths too much I have created a patch that should convert your Euler Angles into Quaternions, giving you back full 3 degrees of rotation.



(This concludes the top part - the calculations for the hung models)

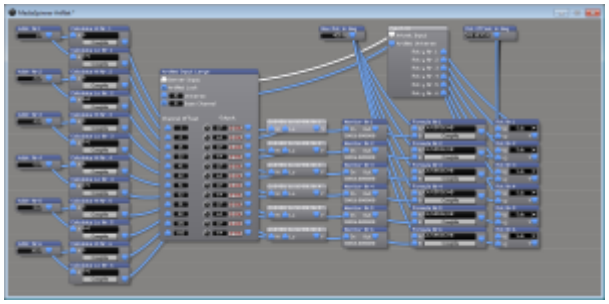
Lower part: Mediaspinners, standing on the ground



This part is pretty much straight-forward: each one of the six screens which are standing in mediaspinners on the ground is represented by its own screen fixtures so that each one can be loaded its own model (again, distinct by the UV mapping), and can be controlled, as per the y roation, via Artnet from the Mediaspinner controller.

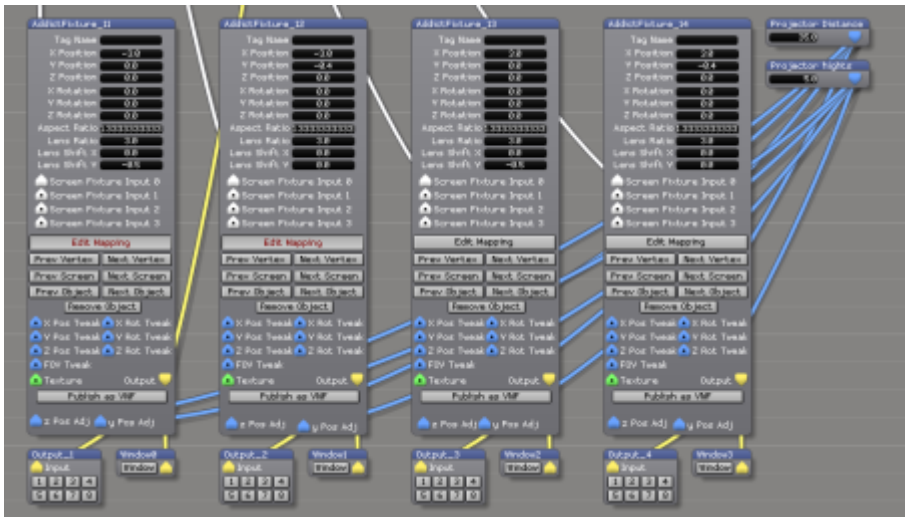
This is done in a little subpatch: MediaSpinner ArtNet

MediaSpinner ArtNet



This subpatch simply allows to set the DMX addresses per mediaspinner - each one takes two DMX channels as 16bit. From the given start address per spinner, the next channel is computed. Then the `ArtNet Input Large` gets us the values, which are then being combined to a 16bit value (0...65,535), which is then normalized (0...1) and multiplied by the full travel (450 degrees). The result is returned, and used as y_rotation parameter `y Angle Adj.` for the respective spinner.

Bottom: the projector outputs



Finally, at the very bottom of the stagepatch, the projectors are defined. Note that again we didn't use softedge, but assigned each screen to a specific projector - see Advanced Output Patch for details (essentially it is disabling `Use All Fixtures` per projector, and patching the screen fixture outputs to the respective inputs).