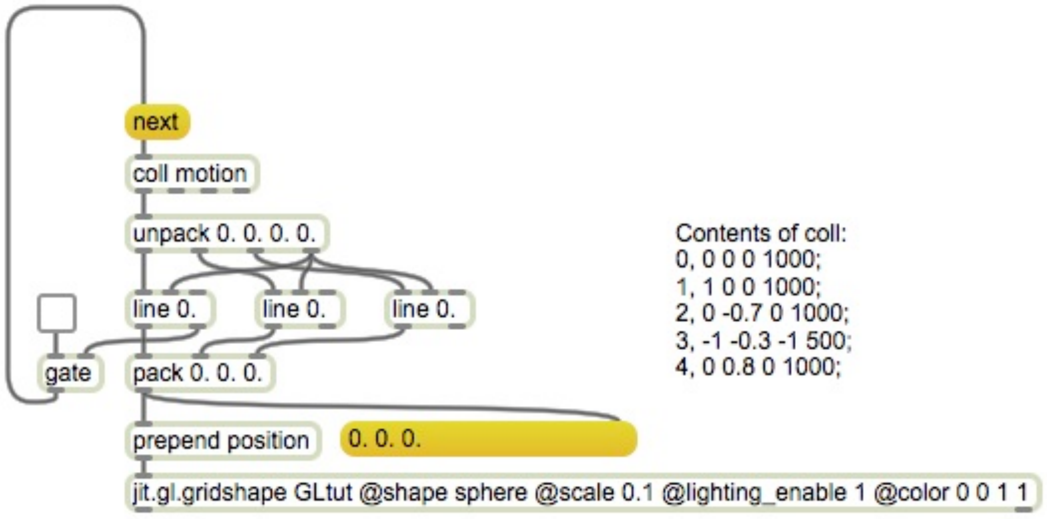
Notes on animation in Jitter

One of the main reasons for using Max/Jitter is the potential for real time or interactive response. Often this means moving things around on a screen. The trick to getting convincing motion is to ensure that the object moves an appropriate distance each frame and follows a graceful path. There are several approaches to this.

Simple Motion with Line and the Like.

Line

If the motion is not complex, position coordinates can be generated as easily with line as anything else.



```
Contents of coll:
0, 0 0 0 1000;
1, 1 0 0 1000;
2, 0 -0.7 0 1000;
3, -1 -0.3 -1 500;
4, 0 0.8 0 1000;
```

Figure 1.

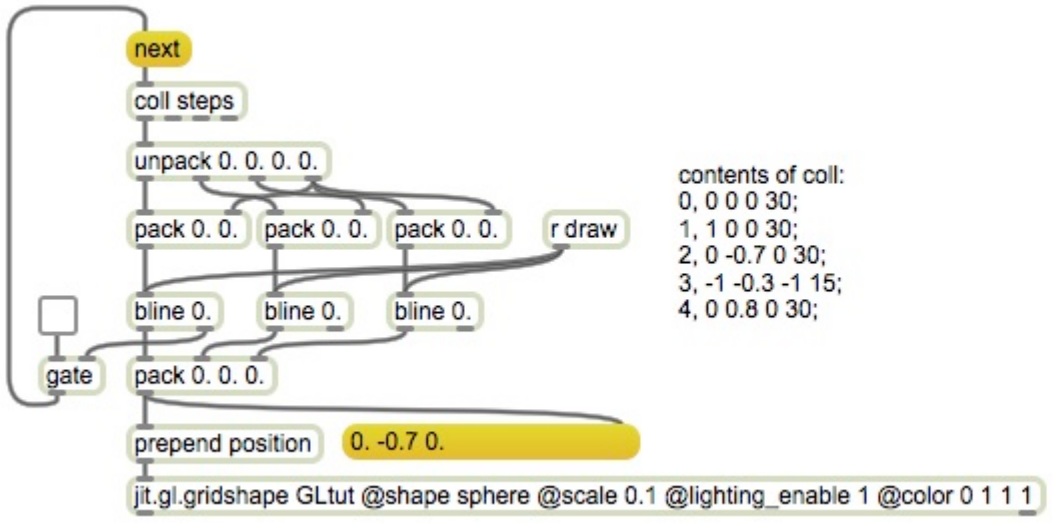
Figure 1[1] moves objects around according to the contents of the coll. Each address in the coll has four values-- X Y Z and the time to take getting there. If the gate is opened, the motion will be continuous if a bit abrupt. The basic principle of this patch can be used in many ways-- for instance clicking on a jit.pwindow could cause some graphic element to move toward the mouse.

Bline

The flaw with the line patch is that when the jitter processing gets heavy, the motion might be erratic. That's because line runs at its own rate regardless of what else is going on. Thus if jitter calculations slow down the system, some outputs of line will be skipped, and the object will make double steps. One solution to this is bline. Bline uses an outside timebase (times are defined in bangs per segment), so it can be driven by the master qmetro as in figure 2. That will make all steps the same size. That is perfect for something that is being recorded by jit.qt.record, because the

[1] All of these patches require the rendering patch shown in figure 2 of the openGl

frames in the resulting movie will be played equally no matter how rough it was in real time.

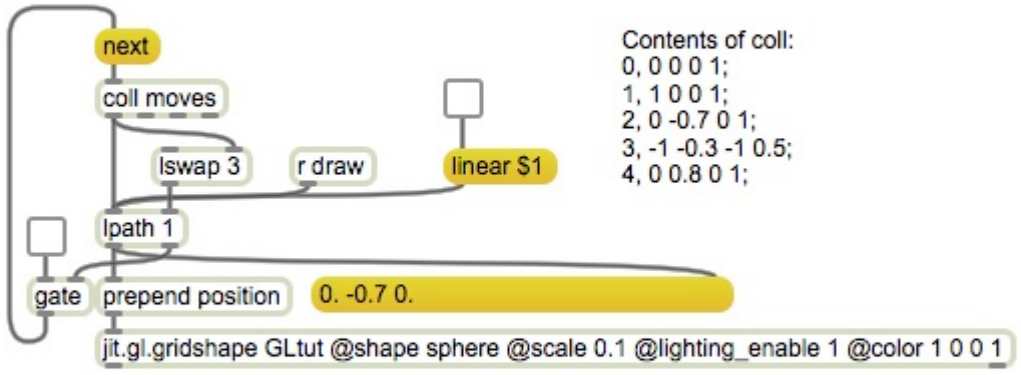


contents of coll:
0, 0 0 0 30;
1, 1 0 0 30;
2, 0 -0.7 0 30;
3, -1 -0.3 -1 15;
4, 0 0.8 0 30;

Figure 2.

Lpath

Of course moving objects are the best way to reveal the effects of jittery frame rates. Bline will not fix this, as objects will pause when the going gets really rough. I added Lpath to the lobjects to help smooth real time display out. Line and bline both divide the path up into equal steps and put out the next value whenever it decides it is time. Lpath reacts to bangs as bline does, but takes the extra step of calculating how far the object should move in the time elapsed. This means real time performance is spot on. (Although recorded motions may be jittery.)


next
coll moves
lswap 3
r draw
linear $1
lpath 1
gate
prepend position
0. -0.7 0.
jit.gl.gridshape GLtut @shape sphere @scale 0.1 @lighting_enable 1 @color 1 0 0 1

Contents of coll:
0, 0 0 0 1;
1, 1 0 0 1;
2, 0 -0.7 0 1;
3, -1 -0.3 -1 0.5;
4, 0 0.8 0 1;

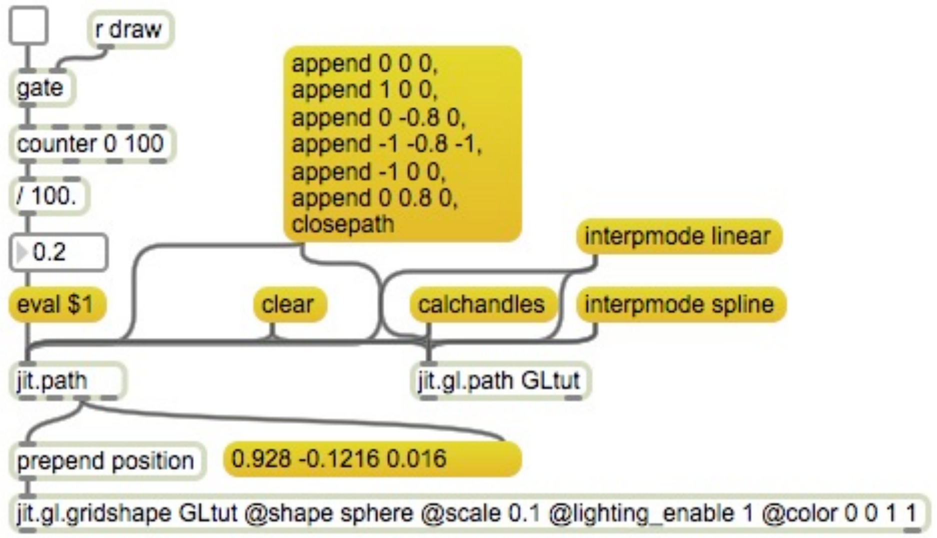
Figure 3.

Figure 3 shows lpath in action. It takes a list as its destination in three dimensions, so only one object is needed. The second inlet (and argument) is the rate of motion in GL units per second. Thus, if an object has further to go, it takes longer to get there. Lpath also has an acceleration feature- instead of blasting out of the gate, it gets up to speed smoothly, and also smoothly slows down as it approaches the target. You can turn this off by setting linear mode.

Motion Control by Vectors

If you want to model motion rather than follow scripted paths, you need to use vectors. This is discussed in the tutorial "Vectors in Jitter".

Jit.Paths



```
append 0 0 0,
append 1 0 0,
append 0 -0.8 0,
append -1 -0.8 -1,
append -1 0 0,
append 0 0.8 0,
closepath
```

```
jit.gl.gridshape GLtut @shape sphere @scale 0.1 @lighting_enable 1 @color 0 0 1 1
```

Figure 4.

The two path objects in jitter use the graphic definition of a path, that is, a complex curve. Jit.path computes a path from a list of vertices. These are built up using the editing commands append and insert. The closepath command adds a segment back to the first vertex. The path may be output as a 2D matrix of vertices and tangents (first outlet) or as a 1D matrix with points interpolated between the vertices. For movement, there is the eval command, which takes an argument in the range 0- 1.0. This produces an interpolated location along the path from the object's third outlet. Figure 4 shows how to use this feature to move objects along the path.

Jit.gl.path will draw a path in a rendering context. It takes the same commands as jit.path and interprets them the same way. Figure 5 shows the path as a line, but there are other options.

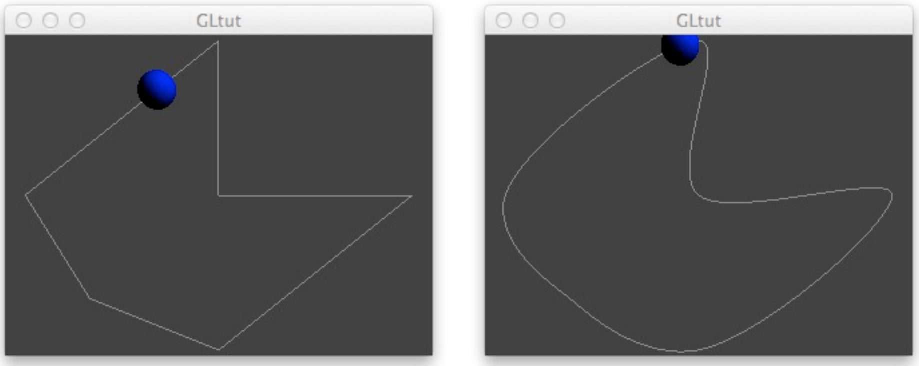
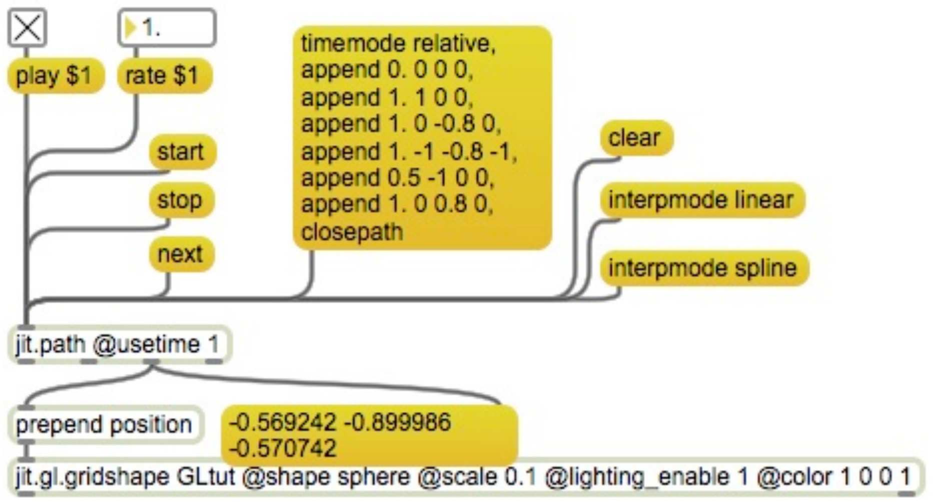


Figure 5.

The path may be straight or curved. To curve a path, set the interpmode to "spline" and send the command calchandles.

The jit.path object can also control movement directly.



```
1.
play $1
rate $1
start
stop
next
timemode relative,
append 0. 0 0 0,
append 1. 1 0 0,
append 1. 0 -0.8 0,
append 1. -1 -0.8 -1,
append 0.5 -1 0 0,
append 1. 0 0.8 0,
closepath
clear
interpmode linear
interpmode spline
jit.path @usetime 1
prepend position
-0.569242 -0.899986
-0.570742
jit.gl.gridshape GLtut @shape sphere @scale 0.1 @lighting_enable 1 @color 1 0 0 1
```

Figure 6.
To get motion out of jit.path, set the attribute @usetime 1. Then each point should begin with a time value. This may be absolute (in which case the points will be sorted in time) or relative, indicating time to reach the point from the previous. To get motion, the play attribute must be set to 1 or the start command sent. Play 0 or stop will prevent position outputs, but the internal time continues to accrue. Thus when you send start again, the object will jump to the current position. To get an object to pause in its tracks, manipulate the rate attribute. 1 is the normal value. A 0 will stop motion, and a negative number will move backwards.

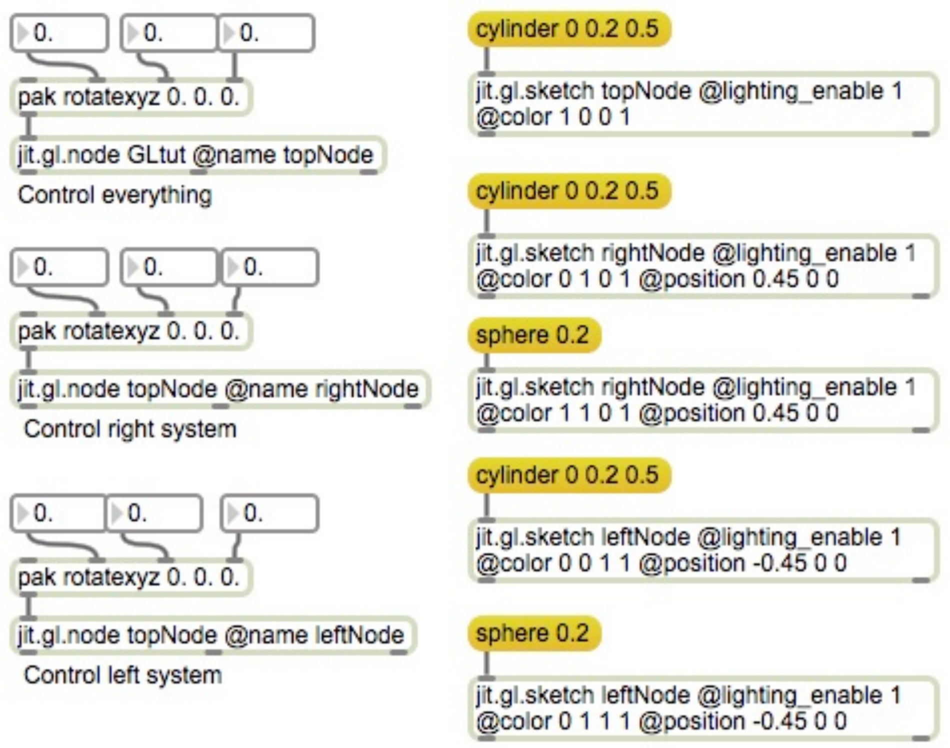
Moving objects in Groups

Jit.gl.node

Sometimes we need to coordinate the movement of many objects to maintain a larger structure. We have learned elsewhere how to move the entire universe, which is properly known as the drawing context[2]. It is also possible to create a hierarchy of sub-contexts and move them independently or in groups. One object that does this is jit.gl.node. Figure 7 shows how to set up a hierarchy of nodes. The top jit.gl.node object has the render context and a @name attribute. The @name attribute creates a subcontext. The next node down uses this name as its context and establishes a sub-sub-context with its own @name attribute. The Max documents call this a parent and child relationship. The drawing objects in the patch can use any of the @names, and will be grouped accordingly. Position, rotate and scale commands are passed down the hierarchy, from parent to child, but commands applied to a child do not affect a parent. Rotate and position commands are additive-- each object has a position and orientation relative to the drawing

[2] You send position or rotate commands to the jit.render object

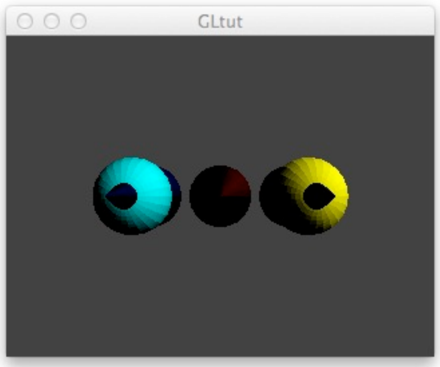
context origin, and position commands to the parent move that origin. Scale is multiplicative. If a parent is scaled to 0.5, children scaled to 0.5 wind up at 0.25 size. Color can be passed down, but overrides child colors.



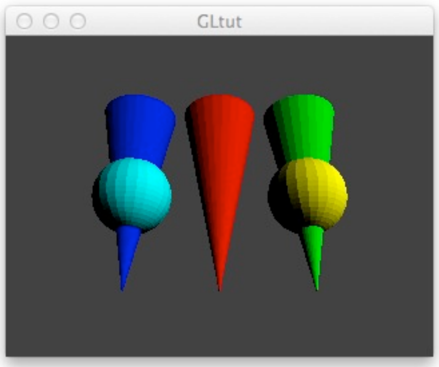
```
0.  0.  0.
pak rotatexyz 0. 0. 0.
jit.gl.node GLtut @name topNode
Control everything

0.  0.  0.
pak rotatexyz 0. 0. 0.
jit.gl.node topNode @name rightNode
Control right system

0.  0.  0.
pak rotatexyz 0. 0. 0.
jit.gl.node topNode @name leftNode
Control left system

cylinder 0 0.2 0.5
jit.gl.sketch topNode @lighting_enable 1
@color 1 0 0 1

cylinder 0 0.2 0.5
jit.gl.sketch rightNode @lighting_enable 1
@color 0 1 0 1 @position 0.45 0 0

sphere 0.2
jit.gl.sketch rightNode @lighting_enable 1
@color 1 1 0 1 @position 0.45 0 0

cylinder 0 0.2 0.5
jit.gl.sketch leftNode @lighting_enable 1
@color 0 0 1 1 @position -0.45 0 0

sphere 0.2
jit.gl.sketch leftNode @lighting_enable 1
@color 0 1 1 1 @position -0.45 0 0
```

Figure 7.

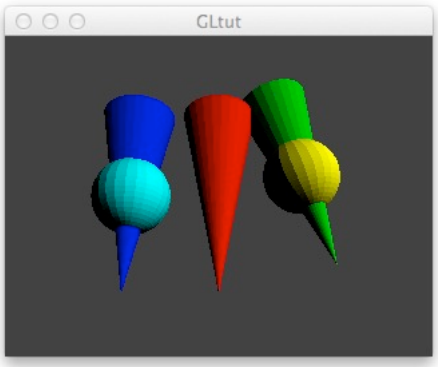
The image created by figure 7 is shown in figure 8, with various rotations applied.



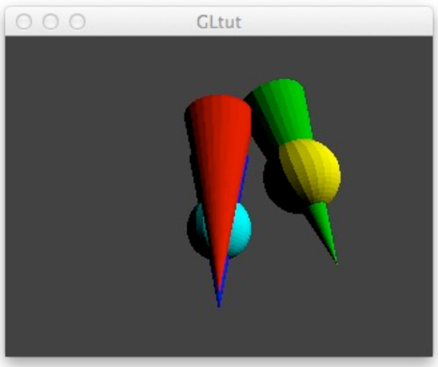
No rotations



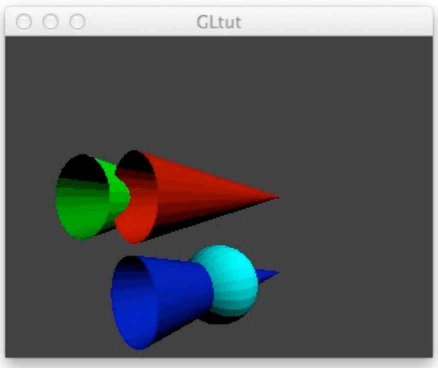
TopNode rotatexyz 64 0 0



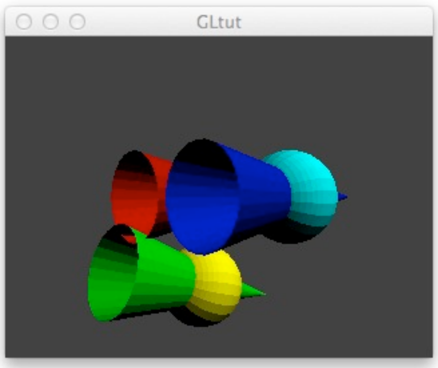
RightNode rotatexyz 0 15 0



LeftNode rotatexyz 0 0 90



TopNode rotatexyz 0 132 0



TopNode rotatexyz 0 132 -90

Figure 8

The red cone belongs only to the topNode, and so follows that rotation. The green cone and yellow sphere are associated with the rightNode, and receives a slight twist in the third frame. The blue cone and sphere are associated with the leftNode and get rotated 90 degrees in the fourth frame. Note that these rotations are relative to the context origin, which is centered for all cases. Any rotation that is needed for an individual object around its own origin would be applied to the object itself. In the final two frames, the topNode is rotated, swinging everything around together.

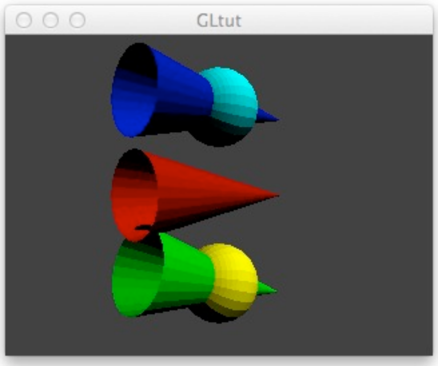


Figure 9.

It is important to carefully consider what objects and contexts should receive position and rotation offsets, as this affects how the objects move together. Figure 9 has the position offsets applied to rightNode and leftNode instead of the individual objects, but has been sent the same series of rotations as figure 8.

Jit.anim.node

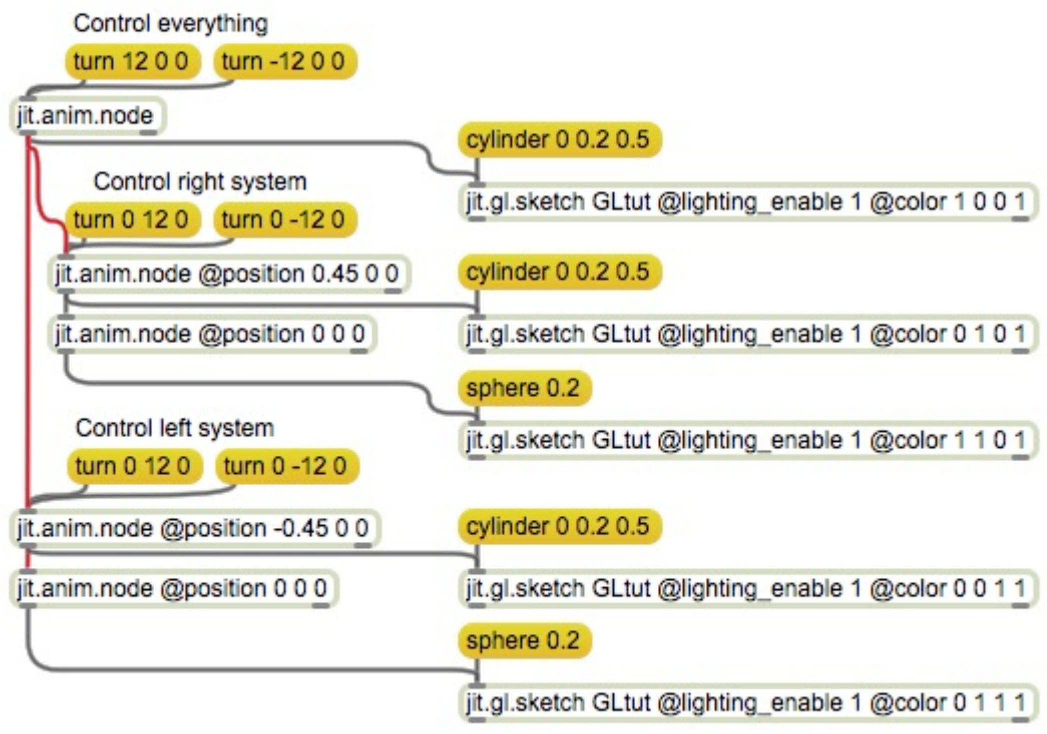

Control everything
turn 12 0 0
turn -12 0 0
jit.anim.node
cylinder 0 0.2 0.5
Control right system
jit.gl.sketch GLtut @lighting_enable 1 @color 1 0 0 1
turn 0 12 0
turn 0 -12 0
jit.anim.node @position 0.45 0 0
cylinder 0 0.2 0.5
jit.anim.node @position 0 0 0
jit.gl.sketch GLtut @lighting_enable 1 @color 0 1 0 1
sphere 0.2
Control left system
jit.gl.sketch GLtut @lighting_enable 1 @color 1 1 0 1
turn 0 12 0
turn 0 -12 0
jit.anim.node @position -0.45 0 0
cylinder 0 0.2 0.5
jit.anim.node @position 0 0 0
jit.gl.sketch GLtut @lighting_enable 1 @color 0 0 1 1
sphere 0.2
jit.gl.sketch GLtut @lighting_enable 1 @color 0 1 1 1

Figure 10.
Jit.anim.node (animation node) provides another system for hierarchical control. Jit.anim.node provides two types of function: node grouping, and translation of simple motion commands into the position and rotatexyz values jit.gl objects need for orientation.

Grouping is established by patch cords. There will be one jit.anim.node object for each jit.gl object we need to control. These connections are made directly as shown in figure 10. Additional connections establish a hierarchy, as shown by the red cords in figure 10.

The control commands recognized by jit.anim.node are **turn**, **move**, and **grow**. These are relative, rather than absolute commands. Turn 12 0 0 is similar to rotatexyz 12 0 0, except repeating the command produces a further rotation. To get back to the original position, turn -12 0 0. Once a node has been turned, any move commands are relative to the object's current orientation.

You can give a node an initial orientation with attributes, and that will be passed on to any children, but a reset command will zero them out, leaving the children behind.

Animated Models

In the jit.gl world, serious animation is done with preconstructed models. Models can be made in any number of applications[3] and opened in jit.gl.model. If the model includes animation rigging (i.e. nodes) jitter can move parts of the model. The jitter helpfiles include an animation model to play with: Seymour the Astroboy.

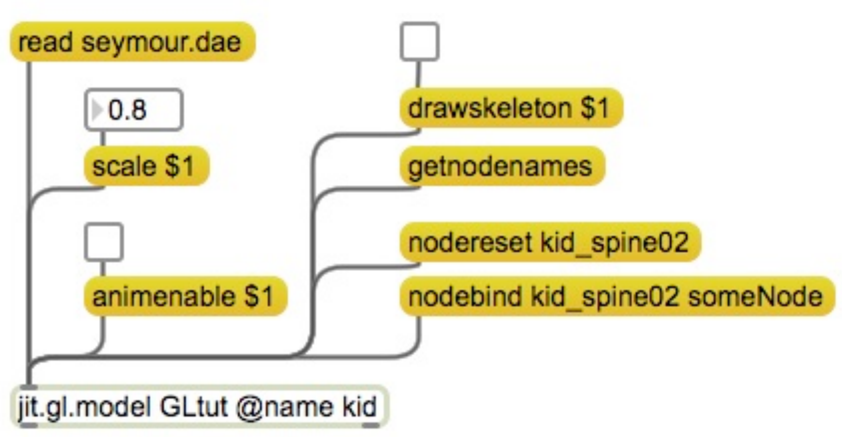


Figure 11.

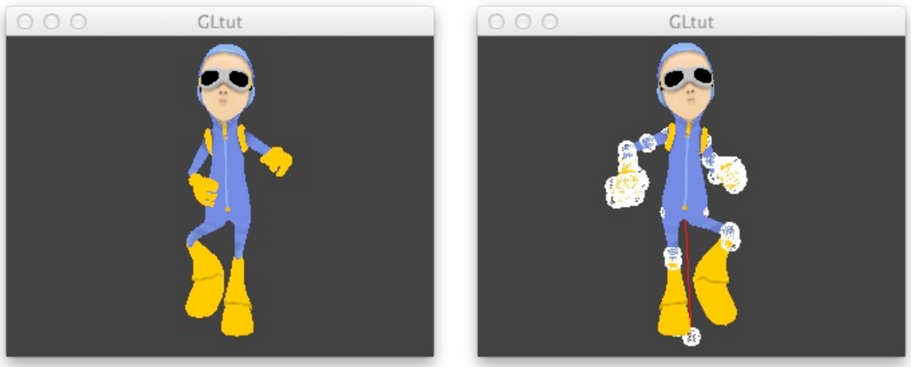
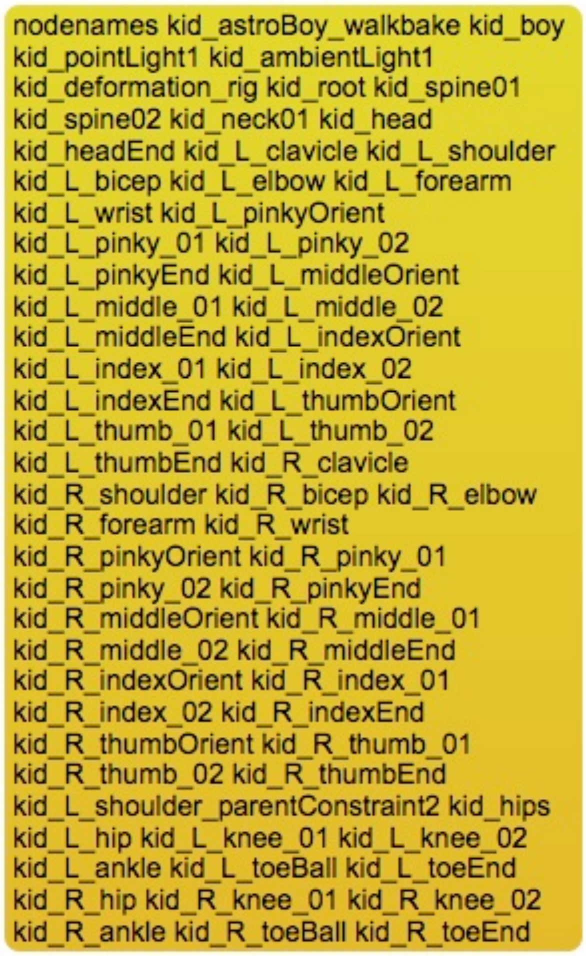


Figure 12.

Seymour is shown in figure 12. He is loaded in the jit.gl.model object, and the command animenable 1 lets him strut his stuff. If you enable drawskeleton, the nodes and bones will be drawn.

The walking animation is built into the dae file. Position and size commands to jit.gl.model will adjust his orientation. You can move Seymour yourself if you discover the appropriate node names. The getnodenames command will produce a listing like figure 13.

[3] You can also find them on the internet at places like OurBricks and TurboSquid.



```
nodenames kid_astroBoy_walkbake kid_boy
kid_pointLight1 kid_ambientLight1
kid_deformation_rig kid_root kid_spine01
kid_spine02 kid_neck01 kid_head
kid_headEnd kid_L_clavicle kid_L_shoulder
kid_L_bicep kid_L_elbow kid_L_forearm
kid_L_wrist kid_L_pinkyOrient
kid_L_pinky_01 kid_L_pinky_02
kid_L_pinkyEnd kid_L_middleOrient
kid_L_middle_01 kid_L_middle_02
kid_L_middleEnd kid_L_indexOrient
kid_L_index_01 kid_L_index_02
kid_L_indexEnd kid_L_thumbOrient
kid_L_thumb_01 kid_L_thumb_02
kid_L_thumbEnd kid_R_clavicle
kid_R_shoulder kid_R_bicep kid_R_elbow
kid_R_forearm kid_R_wrist
kid_R_pinkyOrient kid_R_pinky_01
kid_R_pinky_02 kid_R_pinkyEnd
kid_R_middleOrient kid_R_middle_01
kid_R_middle_02 kid_R_middleEnd
kid_R_indexOrient kid_R_index_01
kid_R_index_02 kid_R_indexEnd
kid_R_thumbOrient kid_R_thumb_01
kid_R_thumb_02 kid_R_thumbEnd
kid_L_shoulder_parentConstraint2 kid_hips
kid_L_hip kid_L_knee_01 kid_L_knee_02
kid_L_ankle kid_L_toeBall kid_L_toeEnd
kid_R_hip kid_R_knee_01 kid_R_knee_02
kid_R_ankle kid_R_toeBall kid_R_toeEnd
```

Figure 13.
The names are dumped from the right outlet of jit.gl.model. You may want to trap them in a umenu instead of a giant list. Note that each node name begins with the name of the jit.gl.model object. You should name your model-- otherwise the names will begin with something unwieldy and impermanent like u002300034.

Once you have the names, you can begin to experiment to discover how the model moves. Nodes are controlled by the jit.anim.nodes object.

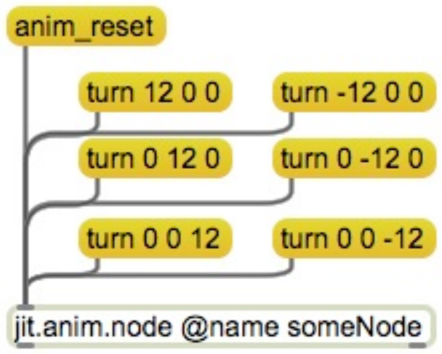


Figure 14.

You need to give a unique name to the jit.anim.node object so you can bind it to one of Seymour's nodes in jit.gl.model. (Binding means to make two entities equal--here, it means messages to jit.anim.node will also be sent to the bound node.) The message "nodebind kid_spine02 someNode will establish a connection between the patch of figure 14 and the kid_spine02 node in jit.gl.model . Having done that, the turn command to someNode will produce the postures in figure 15

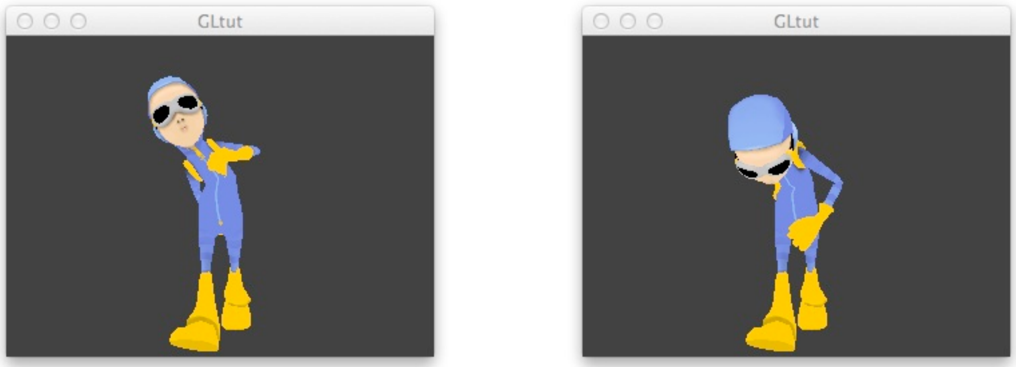


Figure 15.

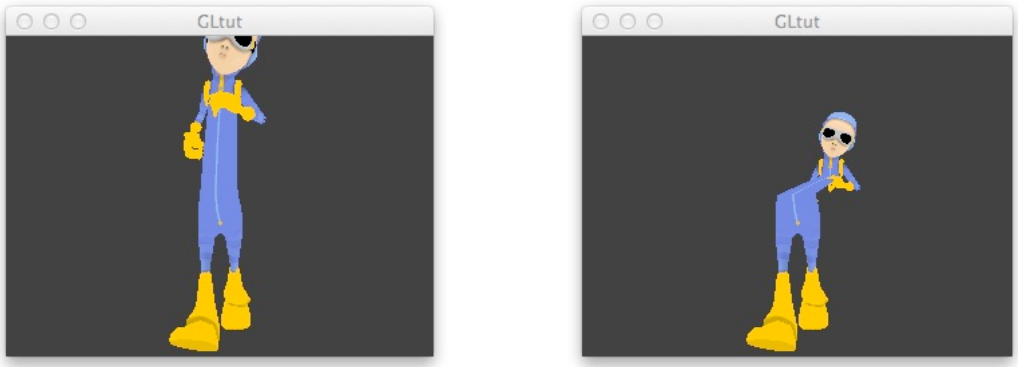


Figure 16.
Figure 16 shows the effect of move commands on this sort of animation. Probably not what is needed, but occasionally useful.

Total control

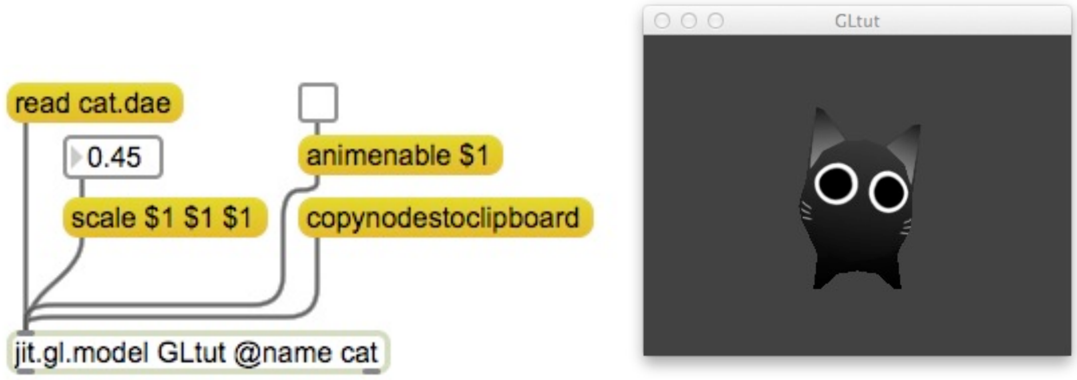
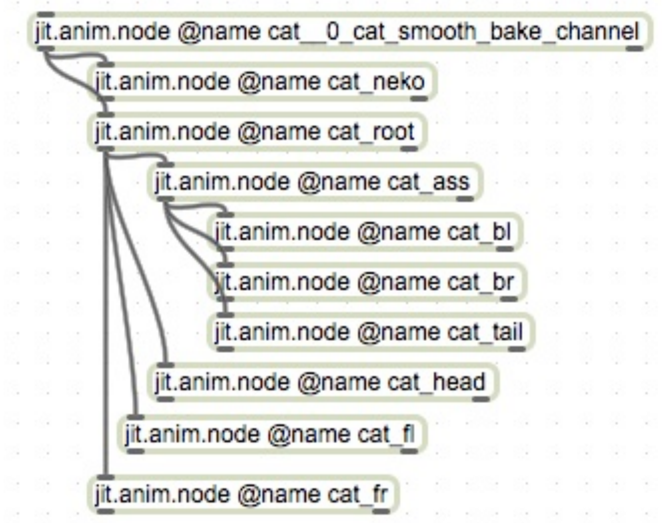


Figure 17.

Figure 17 shows a simpler animation, a cat that bobs its head. We can persuade Max to create a complete set of its nodes with a two step process:
1. Send the message copynodestoclipboard to jit.gl.model
2. Paste into any available window. (Like a subpatcher)

The resulting structure is shown in figure 18>



```
jit.anim.node @name cat__0_cat_smooth_bake_channel
jit.anim.node @name cat_neko
jit.anim.node @name cat_root
jit.anim.node @name cat_ass
jit.anim.node @name cat_bl
jit.anim.node @name cat_br
jit.anim.node @name cat_tail
jit.anim.node @name cat_head
jit.anim.node @name cat_fl
jit.anim.node @name cat_fr
```

Figure 18.
These are all of the nodes in the cat's skeleton. (Seymour's is enormous.) You can send turn messages to any node, and it will control all of the elements attached.

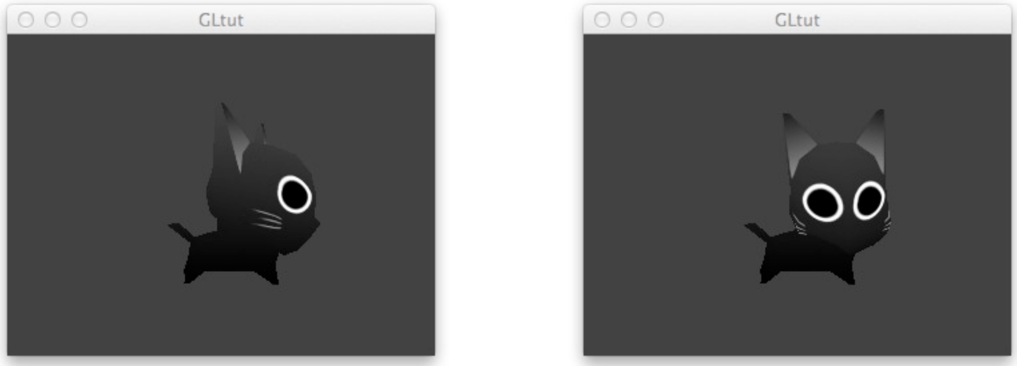


Figure 19.
By applying some turns to the top node, I moved the kitty to a profile view, then with various commands to the cat_head node I could make him nod and twist. Figure 20 shows these modifications:

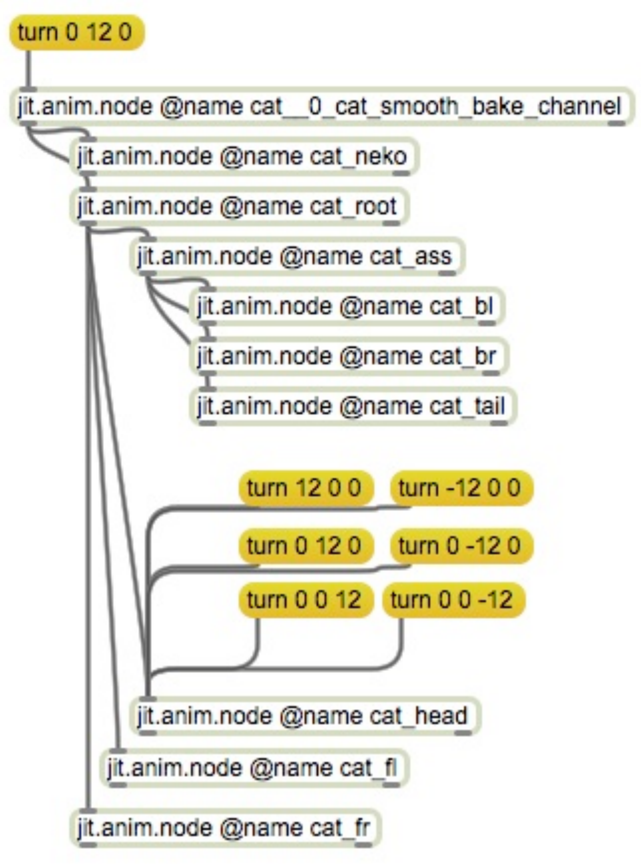

turn 0 12 0
jit.anim.node @name cat__0_cat_smooth_bake_channel
jit.anim.node @name cat_neko
jit.anim.node @name cat_root
jit.anim.node @name cat_ass
jit.anim.node @name cat_bl
jit.anim.node @name cat_br
jit.anim.node @name cat_tail
turn 12 0 0
turn -12 0 0
turn 0 12 0
turn 0 -12 0
turn 0 0 12
turn 0 0 -12
jit.anim.node @name cat_head
jit.anim.node @name cat_fl
jit.anim.node @name cat_fr

Figure 20.

More…..
There are several other objects that are useful in animation:

Jit.anim.drive controls jit.gl objects in a manner similar to jit.anim.node. It does not do hierarchies, but has a wider range of move commands. It also boasts a direct connection to the mouse and keyboard.

Jit.anim.path controls the motion of a jit.gl object like jit.path, but can also include rotation and scale commands in the list of points.

The jit.phys objects allow you to design an environment with gravity, rigid obstacles and collision detection.