Introduction

I love improvisation and, at the same time electronic / computer-based music. Many Ableton Live users will recognize this duality as their own. On the other side I am a visual person, and I always see (seek) a connection between music, shapes color and movement. This personal view and the technical possibility for very thight A/V synchronization are the reasons for V-Module existence.

V-Module is a collection of Max for Live devices for real-time video content creation and processing within Ableton Live. It works on top of Live and is essentially a Live-like interface to Jitter functionalities in Max (along with a few creations of mine).

What can you do with V-Module?

V-Module can be used for video related creation. Although it provides tools for video clip/footage playback, mixing and effect manipulation, where V-Module really shines is on the video "synthesis" side (3D, generative code) and on extreme manipulation (feedback, shading/GPU based effects). Everything, rigorously real time.

As a M4L toolset, V-Module brings things together:

- Jitter's advanced real-time video manipulation
- Jitter's interface to 3D creation (OpenGL, shaders)
- Live automation (clip envelopes, synchronization)
- Parameter automation recording
- Control surface mapping

This tool set is also open and extensible (i.e. it interfaces natively with another great M4L tool for video handling, VIZZable, and through the included documentation it allows you to write your own devices and integrate them in its workflow).

Intellectual Property

V-Module use and Intellectual PropertyV-Module is freely distributed under the Creative Common "No Derivative Works" licence. It is a genuine, self-developed tool, with no particular strings attached: you are completely free to use and modify it as a tool for personal use (media production or performance), even for your professional activities. You are NOT allowed, though, to take any part of it (concepts, algorithms, devices or code) and offer it / commercialize it as an "original" product of your own.

Have fun.

Fabrizio (aka June74)

http://www.fabriziopoce.com/

Table of Contents

Devices and Connectivity

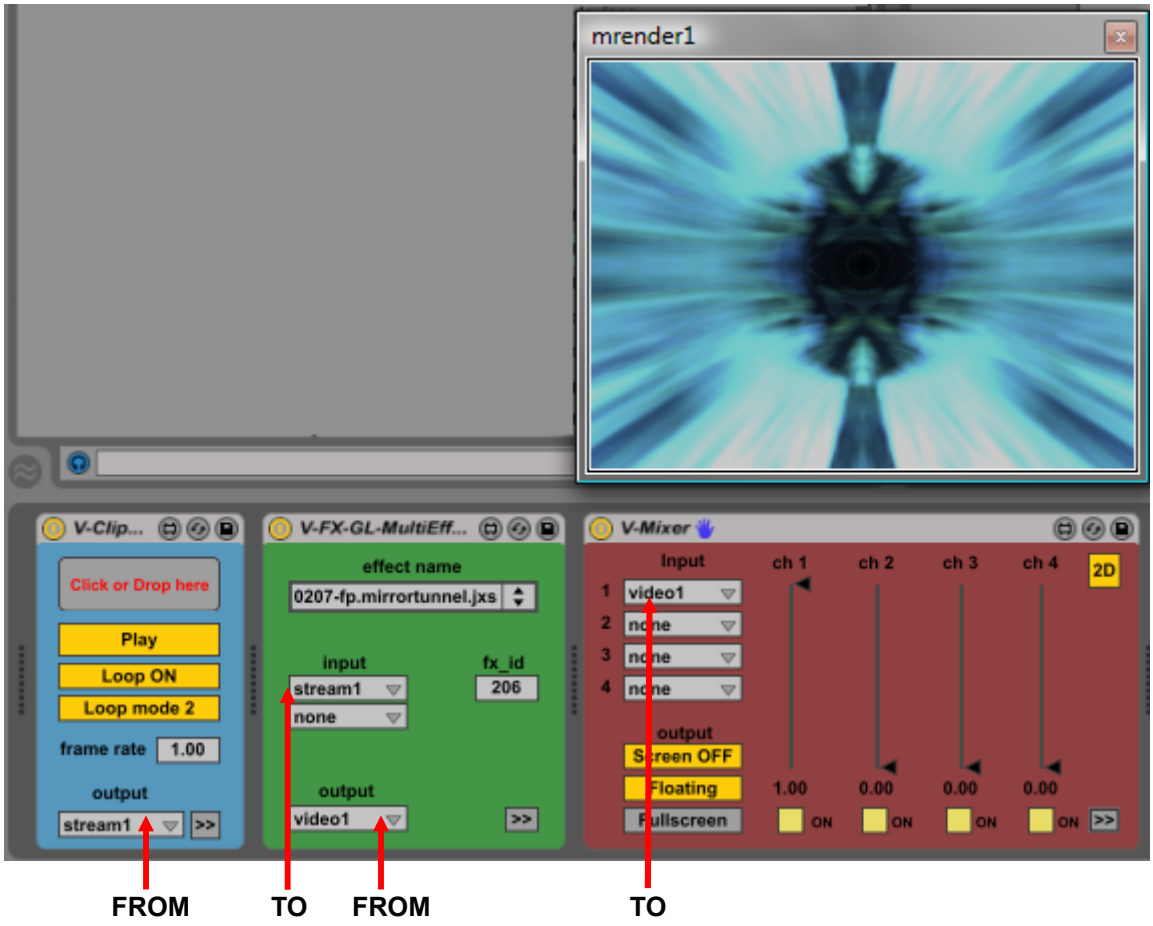
V-Module is composed by a set of modules: devices are relatively compact and perform as atomic building blocks in a flexible chain. The point is: devices can be combined in chains (within Live) without needing to create a new patcher or modify the existing ones in order to interconnect them. Chains can be modified easily, at any time and in real-time. This offers an improvisation environment, where you can play with things in creative ways, without bothering any more (at least ideally) about any programming layer underneath.

V-Module device connectivity

The basic thing you need to understand in order to be able to get something out of V-Module is to catch the concept of video/data *stream bus*. For who is familiar with programming in Max/MSP/Jitter, this is no other than the use of send/receive functionality between patchers, in particular in the context of Jitter matrix I/O (with some extra attention to s/r names and their scope, see use of "---" based s/r for some specific case).

For the general user, unaware of the details, think of a stream bus as a bus channel of "data" in your computer, similar to an audio send in a mixer. A device can put data in this bus and another can get this as input to work on, producing its output and putting this to another stream bus.

Look at the simple example below: in this case a V-Input device sends out its output on a stream bus (named "stream1") to an effect ("V-FX-GL-MultiEffect.amxd"). The latter does the same towards the mixer device (using this time the "video1" bus).



There are a few other types of connection (i.e. in handling 3D graphics), but you should get the point.

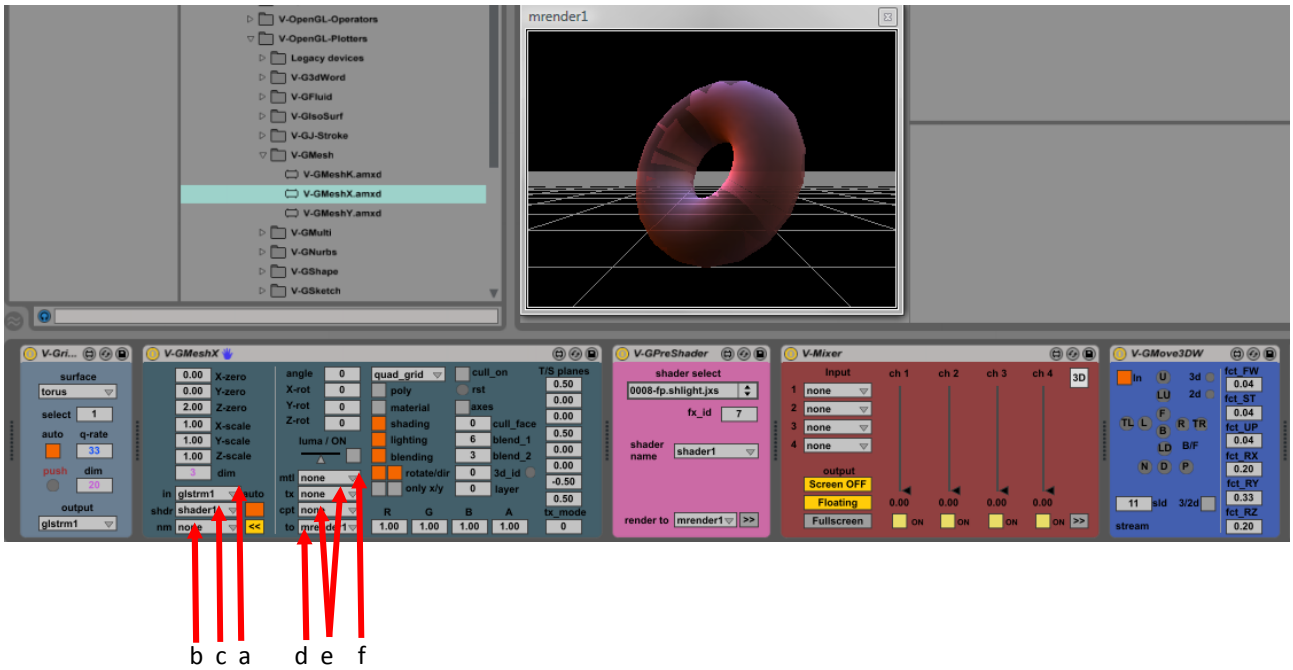
Connectivity: Bus (stream) types

As in Max/MSP/Jitter objects use very different data types (i.e. audio, MIDI, scalars, matrices, lists etc), it is plain that also different data types are involved in V-Module devices. Anyway when interconnection is concerned (thanks to Max flexibility in terms of data conversion) the V-Module user only needs to consider the following stream few basic types:
1) Video clip playback/effects/mixing (directly compatible with VIZZable):

- "**stream**" busses (sometimes named also "video" in the devices) are communication channels used for Jitter's native *RGBA frame matrix format (*a 4 plane matrix of normally char values) communication. Just think of this type of stream as a sequence of video frames expressed as 4 parallel layers of pixels defined in RGBA space, or even more plainly think of it as of a video bus (like an audio bus, but containing video frames). The previous example showed how this video streams are used for video frame routing across the devices.

Note: When working in 2D, even if there is not 3D space composition, underwater a render context will be used (once the V-Mixer is instantiated). This is required to be able to process video frames on the GPU (i.e. Using the "V-FX-GL-MultiEffect", which does effects by loading a pixel shader on the GPU). This approach is required for taking advantage of the GPU power. As an additional reminder, when working in 2D the V-Mixer should be set in 2D mode (toggle on the right corner to "2D").

2) OpenGL/3D stream types:



- "**gstream**" (a) CPU busses are used to communicate spatial information using Jitter's matrix format (the same type of structure as in the previous case) as OpenGL vertex descriptors. A gstream uses *OpenGL 3D matrix formatting,* with at least 3x planes of vertex coordinates (XYZ), with eventually additional planes (up to 13 in total) if other OpenGL parameters are being used. Think of this as a channel going from your CPU to your graphic card, used to communicate spatial information. This channel is not carrying frames, but coordinates and parameters.
- "**object**" (b) names, which can be used by a 3D plotter device (such as V-GMeshX) to identify itself on the GPU and then be picked up for additional composition by another 3D object, usch as the V-GMultiGL device (jit.gl.multiple object).
- "**shader**" (c) names, which refer to GPU (graphic card) shaders, little programs to make material-like

taxellation on the 3D object surfaces (pixel shaders), vertex manipulation (vertex shaders) or both. Shaders for 3D are loaded using for instance the “V-GPreShader.amxd” device.

- "**render**" (d) are names which define the OpenGL rendering context to draw to/from. In other words this informs Max, the CPU and the GPU of the *location* where the results of GPU processing needs to be projected, being this a screen, a window or even location in the CPU centrally-addressed memory (back as a Jitter RBGA matrix) if you do what it is called as “backread” (from GPU to CPU data copy). This allows you to use the power of GPU processing and eventually to be able to process its result once more by the CPU or GPU for a chain. From V-Module 0.9.x this field is auto-populated by default (with context “mrender1”)
- “**texture**” (e) GPU busses which can be used in two ways: as video frames passed to the GPU (as a texture for a 3D/2D creation, i.e. as the “skin” of the object – in the picture above this can be used in field “tx”) or as an alternative destination for rendering (in Jitter terminology a “capture” or render to texture action), useful for real-time effect processing of a 3D scene or object (in the picture this can be used in field “cpt”).
- “**material**” (f) From V-Module 0.9.x (and since Max 6.x has been introduced) this field defines the material used for shading (a combination of textures and shaders, as provided by Cycling74). The material needs to be loaded with the “V-GMaterials.amxd” device.

Notes: When working in 3D the V-Mixer can be set in 3D mode (toggle on the right corner to “3D”), removing the 2D video plane for 2D mixing (if this is required for your usage).

Other bus types:

- "**CVsend**" are very simple busses composed of integer/floating scalar values, which generally change in time. They are multi purpose busses and are generally employed as LFO / modulation signals between devices.

Devices: Categories and purpose

There are a lot of devices I have already built for V-Module. More and more are being added as I get time and ideas for them, while some may be obsoleted and moved in the “Legacy Devices” sub-folders. In order to overview this large set of devices, I found easier to divide them in categories, based on their I/O capabilities. Here are the categories I have come up with.

Traditional 2D processing:

- **V-Input**: open an external media (i.e. a video file), play/adjust/process it and send it as output to one or more video "stream" busses. Devices in this category perform some sort of playback of video clip(s) or open a video camera input.
- **V-Instrument**: algorithmic-based generation of video content which generate output to one or more "stream" busses. These are “creative” devices and each time I learn something new (or find out of interesting work of somebody else which may be integrated in V-Module) I will add a new one. These are somehow the equivalent of like your audio synthesizers plug-ins.
- **V-Effect**: as the name says, they take one or more streams in ingress and do some "effect" (processing) on them giving one or more new "stream" (of the same type of bus) as output. PS: as of today all V-Effects have their processing performed on the GPU (“in hardware”) rather than on the CPU (“in software”). This dramatically enhances quality and fps (frame rate) of you live set.
- **V-Mixer**: a 1x render context + 4x video stream channel mixer, with a built-in per channel RGB filter based on a custom OpenGL shader. It also allows additional OpenGL native operations on video content (such as 3d rotation, selection of 3d primitives, position layout, camera movements, lightening etc.). This is basically “the” main device for terminating all chains in your setup (2D or 3D). Its window would often be your audience performance screen (normally on full-screen) to be projected.
- **V-Output**: straight-forward utilities for additional output windows, real-time measurement of fps, recording to a file, etc.

3D stuff:

- **V-OpenGL-Camera and Position:** this device allow you to move in the 3D space, plus a number of additional actions (like record/activate 3D positions, record/playback flying trajectories and so on).
- **V-OpenGLGenerators**: they create/manage virtual objects as 3D vertex coordinates and parameters. They output a "gstream" OpenGL matrix which can be further processed and eventually sent to the GPU for rendering (i.e. by a V-OpenGLPlotter). This category includes simple Jitter’s native surface generators as well as ad-hoc processing units I created myself based on some mathematics or video-to-3D digitizer (sort of samplers in the video domain). Have fun!
- **V-OpenGL-Operators**: they take one or more "gstream" (3D vertex coordinate matrix) in ingress and allow you to do some calculation/effect on them, generating one or more new "gstream" in output. This allows you to do interesting things such as modulate 3D coordinates with a LFO or audio feed (CVsend), add a Noise generator or mix together several 3D vertex simple surfaces getting out “monsters” in 3D. Again, have fun....
- **V-OpenGLPlotters**: These are the devices which get you to the GPU for 3D drawing and final composition. Many of them need a gstream in input and do for them the 3D plotting and operations (i.e. rotation, position, scale, color, texture). Some of them, though, have an internal 3D vertex generator (either based on algorithms or on the transformation of other media forms such audio, words, video streams, MIDI) and can produce interesting 3D creations. The range of 3D objects cover a wide range: from simple objects (V-GShape) to

complex 3D models (V-ObjShape), to 3D cloning (V-GMulti) and more.

- **V-OpenGL-Processors**: They are the devices doing the shader-based taxellation processing (like assigning a shader or a material to an object surface, or for post-processing a 3D object/scene).

  – The sub-category "V-GShadingEffects" provides 3D surface shading (taxellation). The shaders (loaded on devices such as V-GPreShader.amxd) perform custom lighting, color, vertex manipulation, fog etc.)

  – The sub-category "V-GMaterials" (alternative to the previous one) performs taxellation using Cycling74 new Max 6 material feature.

  – The sub-category "V-GPostProcess-IO" does not do 3D taxellation but, instead, allows post processing of the 3D stuff. It requires first the 3D object/scene to be rendered to a texture (see "capture" option on 3D plotters and V-Mixer), as taken by the V-GPostProcessIN.amxd device, it then allows you to do normal video effect processing (i.e. Using the V-FX-GL-MultiEffect.amxd) and finally posts the results in 3D. It then uses a tranparent video plane on top of the 3D scene (V-GPostProcessOUT.amxd).

- **V-OpenGL-Texture**: Utilities for I/O of textures.

Audio Analysis:

- **V-AuScope**: These are the devices that can get Live's audio (as input) interpret it algorithmically and output, depending on the case, various kinds of stream (video, OpenGL, CVsends etc.). They allow you to have audio modulating one or more of the other domains (video, MIDI, 3D coordinates, whatever….). PS: as they require audio from Live they are not M4L MIDI effects (all all other V-Module devices), but M4L *audio* effects.
- **Au-Control**: analyzes audio and extract information about beat, envelope of frequency content. Useful for modulation (direct via API or via CV-send LFO-like busses)
- **Au-Generators**: analyzes audio and generates 3D "gstream" vertex information based on audio envelope and/or frequency content. Useful (specially if combined through V-OpenGL-Operators) to create audio evolving 3D vertex creatures.
- **Au-Scope**: Provide an oscilloscope like source both in 3D and 2D bus formats

CV and Modulation Utilities

- **CV-Modulation**: utilities which can create and/or operate on CVsends. They are comparable to your synth's LFO waveform generators or CV input and include actions such as "produce MIDI to/from CVsend" or "create an LFO as a CVsend" and even more complex manipulations. They allow you to mix and connect different dimensions together and get "multi-media" effects out of V-Module.
- **CV-CV2External**: From CV to external means such as MIDI or OSC.
- **CV-External2CV**: From external means (such as MIDI and OSC) to CV-sends
- **CV-Operators**: Allows to mathematically operate on CV-sends
- **Conversion Tools**: Utilities for matrix convertion

Some Tips

Tips for optimizing performance (all platforms):

- Set Ableton Live audio buffer size to the lowest possible value for your system and audio applications (Preferences -> Audio -> Hardware Setup)
- If you have a GPU (nVidia, AMD/ATI, etc) use GPU based processing whenever possible.
- If you use video clips playback use either ProRes, Motion JPEG or Photo JPEG codec preferably with MOV or AVI containers.
- In Live 8 correct behavior and best performance currently achieved using Max5 as M4L engine (Ableton 8.2.6 and higher, with Max 5.1.9). Max 6.0 is not recommended in combination with Live 8. In Live 9 use Max 6.1.x or later.

Tips for optimizing performance (Mac users only):

- Mac users can limit video resourse contention between Live's GUI and M4L visual applications, *by setting the Live GUI in the background*. In fact doing video in Max for Live means Live's GUI and Max for Live will fight for the same resource: OpenGL processing power from the GPU. Setting Live's GUI in background reduces this effect.
- This limitation is not applicable to Windows users, as in this case OpenGL is not used by Live's GUI (the GUI uses DirectX on a Windows environment), leaving OpenGL resources free to M4L, resulting in a somehow more predictable event scheduling.

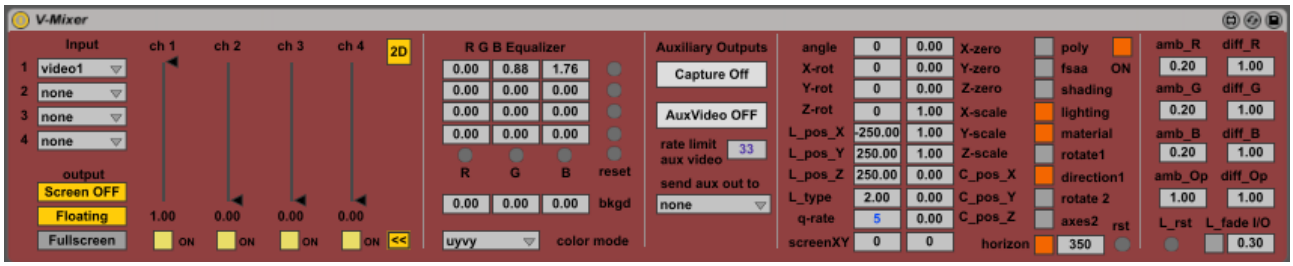
Various Tips:

- Use a different MIDI track for each sub-chain and this helps finding things (as you can name the track) and allows you better automation (by using dummy clips and clip envelopes)
- Try experimenting with clip envelopes in (MIDI) dummy clips. They can automate many actions for you.
- Try recording envelopes using both clip recording (Live 9 only).
- Watch out for bus stream collisions and loops
- Play with CVsend…… it’s fun

Essential Device list

Here you can find details on the *most commonly used* V-Module devices. As the device list is quite long I do not have the possibility to explain all the details on all the plugins. Anyway, any device falls in one of the categories initially introduced: so by association with similar devices in the same category you should be able to get an understanding device connectivity I/O options and by experimentation on the parameters exposed (.....) you can relatively easily get an hold on their specific purpose (and by the way, you can always hit the edit button and look inside.....)

V-Mixer
The heart of any V-Module setup. An OpenGL-based 1x render + 4x video stream channel mixer, with per channel RGB filter and a number of OpenGL 3D controls. It is based on a custom GLSL shader I developed.



4-ch video stream mixer section:

- input1 = input 1 video stream
- input2 = input 2 video stream
- input3 = input 3 video stream
- input4 = input 4 video stream
- input color mode = argb | uyvy
- output format = output format (should match input format)
- fullscreen = fullscreen selector/button
- ch1 = mixer gain/slider for input 1
- ch2 = mixer gain/slider for input 2
- ch3 = mixer gain/slider for input 3
- ch4 = mixer gain/slider for input 4
- ON1 = mute (on/off) for input 1
- ON2 = mute (on/off) for input 2
- ON3 = mute (on/off) for input 3
- ON4 = mute (on/off) for input 4

RGB filter and texture redirection section:

- RGB filter row1/col1 = Red filter for ch1
- RGB filter row1/col2 = Green filter for ch1
- RGB filter row1/col3 = Blue filter for ch1
- RGB filter row2/col1 = Red filter for ch2
- RGB filter row2/col2 = Green filter for ch2
- RGB filter row2/col3 = Blue filter for ch2
- RGB filter row3/col1 = Red filter for ch3
- RGB filter row3/col2 = Green filter for ch3
- RGB filter row3/col3 = Blue filter for ch3
- RGB filter row4/col1 = Red filter for ch4

- RGB filter row4/col2 = Green filter for ch4
- RGB filter row4/col3 = Blue filter for ch4
- reset row1/col4 = reset RGB for ch1
- reset row2/col4 = reset RGB for ch2
- reset row3/col4 = reset RGB for ch3
- reset row4/col4 = reset RGB for ch4
- reset raw5/col1 (R) = reset all channel Red filters
- reset raw5/col2 (G) = reset all channel Green filters
- reset raw5/col1 (B) = reset all channel Blue filters
- reset row5/col4 (reset) = reset every filter
- bgrd RGB row6/col1 = OpenGL 3D Background Red component
- bgrd RGB row6/col2 = OpenGL 3D Background Green component
- bgrd RGB row6/col3 = OpenGL 3D Background Blue component

3D Rotation, Lighting, Position and Camera section:

- angle = angle of 3d rotation
- X-rot = component of 3d rotation around axis X
- Y-rot = component of 3d rotation around axis Y
- Z-rot = component of 3d rotation around axis Z
- L_pos_X = position of light source coordinate X
- L_pos_Y = position of light source coordinate Y
- L_pos_Z = position of light source coordinate Z
- L_type = type of light source
- q-rate = output frame rate
- X-zero = center position offset on direction X
- Y-zero = center position offset on direction Y
- Z-zero = center position offset on direction Z
- X-scale = objects scaling on direction X
- Y-scale = objects scaling on direction Y
- Z-scale = objects scaling on direction Z
- C_pos_X = position of camera coordinate X
- C_pos_Y = position of camera coordinate Y
- C_pos_Z = position of camera coordinate Z
- ON = device processing ON/OFF
- float = set the floating flag ON (device does not move behind other windows).
- rst = reset any 3D parameter to default

Rendering section:

- poly = ON draws empty primitives, OFF primitive in texture/material
- material = color components of texture/material are supported
- shading = applies shading (gradual attenuation) to geometric surface curves
- lighting = applies lighting effect based on light source position / type
- axes1 = show main XYZ axes
- rotate1 = auto-rotate mode 1 (automatic, no handle action)
- direction1 = auto-rotate direction (clockwise on/off)
- axes2 = show XYZ rotation2 spheres
- rotate2 = auto-rotate mode 2 (endless if triggered by handle action)

2D/3D toggle: toggles between 2D mode (video plane for mixing) and pure 3D mode (video plane for mixing removed).

V-ClipPlayer.amxd
A video clip player. As all other V-Input devices of this type, i*t requires Quick Time to be installed* and offers response to MIDI notes input (note pitch defines time offset in playback) as well as control for looping and frame rate. Available parameters:

- read area = clip selection (file browser / drag'n'drop)
- p/p = play / pause video
- loop = loop entire clip
- loop2 = loop bounce entire clip (reverse, mode 2)
- frame rate = speed of playback (1.0 = original, negative plays backwards)
- q-rate = output frame rate (video stream clocking, in ms)
- format = frame format (picture dimensions)
- channel = if coupled to MIDI controller CC channels (see pre-mapped CC in patch)
- output = stream bus (video, RGBA frames)

V-FolderPlayer.amxd
A video "folder" clip player. It can be piloted using MIDI notes input (note pitch defines which clip is to be played, starting from C0 for the first clip in the folder) as well as control for looping and frame rate. *It requires Quick Time to be installed.*

V-Cam
Grab video from a web cam or DV device:

- open = open camera input
- clear = clear camera input
- menu1 = device selection
- menu2 = mode selection
- menu3 = camera quality selection
- number_box1 = frame left min edge
- number_box2 = frame right min edge
- number_box3 = frame left max edge
- number_box4 = frame right max edge
- number_box5 = frame up min edge
- number_box6 = frame down min edge
- number_box7 = frame up max edge
- number_box8 = frame down max edge
- button1 = activate edge left/right
- button3 = activate edge up/down
- q-rate = output frame rate
- export = not_supported
- setting = open camera driver settings
- output = output stream

Note: V-Cam has a built in mechanism which discovers the system OS type (windows or mac) and auto detects if jit.qt.grab or jit.dx.grab should be used. *It requires Quick Time to be installed.*

V-ClipAudioPlayer.amxd
This device, functionally very similar to the V-ClipPlayer.amxd, adds audio support for video clips with integrated audio. Please be aware that an audio sample will be automatically extracted from the video clip (see V-VideoAudioRack notes concerning sample location). Important: this is an "instrument" device. *It requires Quick Time to be installed.*

V-SlabGen
This device is capable of hosting shaders for generative, algorithmic video generation. By default it load the OpenGL shaders I developed myself for hardware based fractal generation. As the video stream is generated in hardware by your graphic card (GPU), quality results can be astounding (full HD 1080p included). You need a graphic card with OpenGL 1.20 support.

V-Boids
A fluid simulator (moving “balls” within your screen, each one having its own simulated physical characteristics) implemented in Java for high performance.

V-J-FluidParticles.amxd
A fluid simulator (creating turbolence in a fluid) implemented in Java for high performance. Requires CV-input for the turbolence trigger (X/Y).

V-Lcd.amxd
A simple device which allows you to write (or draw) using the mouse and keyboard. You need to click on the device GUI in Live to be able to enter text.

V-FX-GL-MultiEffect.amxd
This is a flexible OpenGL multi effect based on Jitter's jit.gl.slab, capable of automatically importing all the V-Modules shaders (including Cycling74 shaders, third party (see licencing) and original shader written by myself). The effect menu/selector is automatically populated when the device is dropped in the Live Set, listing all possible / available / found shaders in the path. The device automatically maps up to 16x parameters, along with their name. If a parameter is shown as “none”, the parameter is not active.

PS: depending on your hardware shaders may compile or not (read, work or not). This is something which depends on the system hardware capabilities. If you get a black (or white) screen, the shader has probably failed to compile (if you really want to know, you could verify this by opening the Max editor for the patch and looking at the Max window for error logs).

Parameters:

- input = input video stream
- input color mode = argb | uyvy
- effect = selector for the applied shader type. Any shader available in your Max path should be selectable
- q-rate = output frame rate
- output format = output format (should match input format)
- out = output video stream
- ON = device processing ON/OFF
- PARAMETER BANK of 16x flexible float parameters:
    - Once an effect/shader is selected up to 16x parameters are automatically presented, along with their name. If a parameter is shown as “none”, the parameter is not active.
    - rst = reset all parameter values to default (0,00)
- bck = use backread technique.
    - Set it on if the next device requires a RGBA Jitter video stream (i.e. another V-Effect NOT type GL)
    - Set it off if the next device is an OpenGL device (i.e. a V-Effect type GL or V-Mixer).

V-FX-GL-MultiEffectChain.amxd
Similar to the V-FX-GL-MultiEffect.amxd, it chains three slabs and llows flexible routing, including feedback. The three shaders can be selected and manipulated according to the same logic as in V-FX-GL-MultiEffect.amxd. The matrix / network allows you to route I/O across these slab's / processing units. A dot is a connection between output and input.

V-FX-Looper.amxd
A real-time video looper, similar to Live's Looper.

V-Xconnect
A simple cross-connect if you need to quickly duplicate outputs or if you like to quickly switch from one setup to another. Thought for integration with launchpad or similar. See V-FX-Mosaic4 for an example.

V-Meter
A simple utility which visually shows usage of video streams.

V-Fps
An envelope to Jitters jit.fps object. It is your friend to keep an eye on frame rate and quality. The only parameter it requires is the video stream you wish to monitor.

V-NetSend
Send a video stream (Jitter matrix) via TCP/IP to another host. See also V-NetReceive.

- source = select the input stream bus (RGBA frames)
- select destination = IP address of the target host (you need to open the patch to change it)
- type destination = alternative to menu selector above.
- toggle = shows if a TCP/IP session has been established

V-NetStream
Send a video stream (Jitter matrix) via RTSP/UDP/ IP to another host. Must be coupled to a RTSP client (i.e. VLC).

- source = select the input stream bus (RGBA frames)
- codec = video codec used before packetizing and sending the video
- format = video format used when coding
- select destination = IP address of the target host (you need to open the patch to change it)
- type destination = alternative to menu selector above.
- start/stop buttons = activate / deactivate stream
- debug bar = shows the RTSP URL (useful for configuring client)
- unicast = toggle ON activate network unicast mode (recommended). Broacast bode (unicast toggle = OFF) useful if more hosts must receive stream simultaneously
- matrix file = currently work in progress / not implemented fully

V-Preview
Simple patcher with a “pwindow for stream preview. Useful as a preview non-visible secondary screen.

- source = select the input stream bus (RGBA frames)

Note: this is a CPU based device. It will therefore use resources and potentially affect the global fps and quality of your entire A/V Live set. In other words use these little screen with care and if possible do not use them at all!

V-Rec
Record to disk a stream video. Please be aware that frame rate (fps) may be heavily influenced by using this device. This is because recording to disk is a CPU intensive process (interrupting the CPU at very high rates).

If you really need to use it, try to:

- Optimize codec options (my experience is that “animation” appears to give a reasonable compromise in quality and achieved fps).
- Set the “target” parameter fps to the average value measured by a V-fps device monitoring the same stream source tested while trying recording (do a test recording with the V-Fps device, see the achieved fps on average. Before starting the actual final recording, remove the V-Fps device).
- Disk speed influences frame rate dramatically. If you can use a solution emulating a “ram-disk” (memory used as a disk) or a SSD disk. This will improve quality and fps.
- Obviously an external device provides the best possible results – no load on your system at all! See comments I left at the end of this document (about xVGA, DVI or HDMI recorders/grabbers).

Parameters:

- image = captures a single frame as JPG (file browser pop’s up for destination)
- record = starts/stops recording (file browser pop’s up for destination)
- source = select the input stream bus (RGBA frames)
- codec = video codec used before packetizing and sending the video
- quality = video codec setting for target quality
- format = video format used when coding

Alternatives: You can achieve far better results in performance by following one of the following options.

- On Max, use Syphon (supported on V-Mixer)
- On Windows use Fraps
- Use a hardwate HDMI grabber/recorder (such as Matrox MXO MINI)

3D devices / Camera and Position

V-GMove3DW, V-GMove3DWPad and V-GMove3DWJoy
A set of devices for viewer and camera position movement in the 3D space. The device as two purposes:

- Control and move the camera and viewer position as a "add-on" to the V-Mixer
- Provide the viewer coordinates to other devices in V-Module (i.e. the V-FX-GL-Proximity)

The different versions support slightly different sets of controls:

- V-GMove3DW is the standard device with a button for each direction. This are the same controls specified for the V-GRender3DW1 device. The V-GMove3DW device though does not render anything directly, but works as an add-on to render devices such as the V-Mixer.
- V-GMove3DWPad, exactly the same as the V-GMove3DW but with built in support for a HID Gamepad (two joysticks + 10 buttons)
- V-GMove3DWJoy, exactly the same as the V-GMove3DW but with built in support for a HID Joystick (with 3 axes and 3 rotations, plus 10 buttons)

Note: the current version includes non-Java and Java device versions. The non-Java versions do not have any specific additional requirement and have the suffix "-o" in their name. The Java versions (same name, no suffix) are more precise and less CPU-intensive, but require Java to be installed.

3D devices / Generators

V-GridShape
A V-OpenGLGenerator based on Jitter's jit.gl.gridshape simple surface generator. Produces vertex geometry (as a gstream output) and requires a V-OpenGLPlotter for performing rendering to a drawing context. Parameters:

- surface = type of object to be generated (sphere, cube, torus, plane, etc.)
- auto = activate continuous gstream output (useful if you need to do vertex processing)
- push = one time send for gstream output
- q-rate = rate of the gstream output
- dim = number of 3D vertex used by gstream (resolution / quality)

V-DigiPlay
A V-OpenGLGenerator which accepts a RGBA video stream in input and transforms the RGBA information in 3D coordinates. This magic is based on luminosity levels (relative distance in the source image, a similar process to the hardware produced by Rutt Etra in the 70's – search on the Internet for their beautiful video synthesizer hardware). As this device produces vertex geometry, it requires a V-OpenGLPlotter for rendering it to a context.

- source = a video stream which is used as input to the algorithm
- q-rate = clocking of the generated gstream (3D OpenGl matrix output)
- dim = resolution of the generated gstream (3D OpenGl matrix output)
- displace = a for of "gain" which amplifies relative luminosity distances in source video
- sliding = a form of time filter which smoothes output by carrying a number of previous samples
- base surface = a base surface used to generate the digitized effect on top.
- output = the gstream (3D OpenGL) output bus
- ON = device ON-OFF
- Push = in case device is OFF, to test output for a single frame

V-ObjShape
A V-OpenGLGenerator which can read a .OBJ file with 3D vertex definition, generate a gstream (3D OpenGL spatial matrix) for further processing and/or directly send OpenGL instruction for rendering in a drawing context.

- angle = angle of 3d rotation
- X-rot = component of 3d rotation around axis X
- Y-rot = component of 3d rotation around axis Y
- Z-rot = component of 3d rotation around axis Z
- q-rate = output frame rate
- X-zero = center position offset on direction X
- Y-zero = center position offset on direction Y
- Z-zero = center position offset on direction Z
- X-scale = objects scaling on direction X
- Y-scale = objects scaling on direction Y
- Z-scale = objects scaling on direction Z
- material = mode applied to material in shading
- rotate = auto-rotate mode 1 (automatic, no handle action)
- direction = auto-rotate direction (clockwise on/off)
- shading = applies shading (gradual attenuation) to geometric surface curves
- lighting = applies lighting effect based on light source position / type
- R = Red component gain
- G = Green component gain
- B = Blue component gain
- A = Alpha channel gain
- rst = reset any 3D parameter to default

V-GCoord
A flexible V-OpenGLGenerator which uses mathematical formulas to create vertex coordinates. Requires a V-OpenGLPlotter for rendering to a context.

- x-coord = select formula for X component by index number (formula on the menu next)
- y-coord = select formula for Y component by index number (formula on the menu next)
- z-coord = select formula for Z component by index number (formula on the menu next)
- q-rate = clocking of the generated gstream (3D OpenGl matrix output)
- auto = push output continuously (allows mutation by formula change)
- dim = resolution of the generated gstream (3D OpenGl matrix output)
- push = in case device is OFF, to test output for a single frame
- output = the gstream (3D OpenGL) output bus
- custom parameter for the applied formula (generally a gain applied to the function).
- custom parameter called “fine”, useful for fine tuning of the generative functions (as a 0. to 1.0 product gain of the generated values).

V-GDensity
Generates a single plane square density (in the form of a gstream) based on the distribution of a sphere within the Cartesian (cubical) space. Each dimension can be modified in its zero-centre coordinate and for its radius (changed by the “size’ parameter).

This device is useful if coupled to the V-GVolume device, eventually in combination with the V-FX-GDensityOp operator (which allows several densities to be manipulated together). The V-GVolume device though is not limited to one-plane dimensional matrix input and can accept any matrix in input. Only the first plane, though, will be used.

3D devices / V-OpenGL-Operators and Vertex Processors (for gstreams)

V-FX-GLOp and V-FX-GLOpScalar
The V-GX-GLOp device enables simple mathematical processing on a pair of OpenGL vertex 3D input matrices. Useful for creating evolutionary geometric forms based on “combinations” of objects and eventually coordinates produces by alternative media (such as audio). The V-FX-GLOpScalar is a device with only one input being a gstream, the second operator being a scalar.

V-FX-GLSlide
Applies a low-pass-filter like effect to the input, practically resulting in a smoothed output.

V-FX-Fade
Useful for morphing between two object coordinates, using two gstream sets as input and generating a new gstream with output (the xfade between the inputs). Done in time, allows object morphing.

V-FX-GLProximity
Evaluates the current viewer (camera) position in space (as defined by the 3D render devices, in other words this is the position of the viewer in space, i.e. in the V-Mixer) against a given gstream (coordinate set). If the viewer position is in “proximity” of an area in the gstream matrix, a 1 is generated at the same coordinate place. The result is a new gstream with 1’s moving around following the viewer movements in space.

This device has been designed for use in conjunction with the V-GMulti device: the V-FX-Proximity gstream input would be set as the same one used in V-GMulti for the “position matrix”. The V-FX-Proximity output can be used as one of the other V-GMulti matrix inputs (i.e. rotation, scale or color matrices). For example if used as the rotation matrix input of the same V-GMulti, moving the viewer in spaces causes the clones to start rotating when approached, stopping when moving away from them.

V-FX-GLNoiser
A device adding random noise to an input gstream.

3D devices / Plotters

V-GMesh
A useful and versatile OpenGL Plotter, which accepts a gstream (an OpenGL 3D vertex matrix), allows many advanced OpenGL functionalities and sends the instructions to the GPU drawing context.

The following options (common to practically all OpenGL Plotter, Shaders and Renderers devices in V-Module) summarize the available capabilities:

- angle = angle of 3d rotation
- X-rot = component of 3d rotation around axis X
- Y-rot = component of 3d rotation around axis Y
- Z-rot = component of 3d rotation around axis Z
- q-rate = output frame rate
- X-zero = center position offset on direction X
- Y-zero = center position offset on direction Y
- Z-zero = center position offset on direction Z
- X-scale = objects scaling on direction X
- Y-scale = objects scaling on direction Y
- Z-scale = objects scaling on direction Z
- material = mode applied to material in shading
- text_map = type of texture mapping
- rotate = auto-rotate mode 1 (automatic, no handle action)
- direction = auto-rotate direction (clockwise on/off)
- shading = applies shading (gradual attenuation) to geometric surface curves
- lighting = applies lighting effect based on light source position / type
- R = Red component gain
- G = Green component gain
- B = Blue component gain
- A = Alpha channel gain
- luma: set luminosity anount (RGBA all equal)
- rst = reset any 3D parameter to default

V-GVideoPlane
An OpenGL Plotter which accepts a video stream as input and projects it on a 3D plane in space.

V-ObjShape
An OpenGL Plotter which can load .obj 3D definition files and plot them in space.

V-GridShape
Identical to V-GridShape OpenGL Generator, but with direct drawing to a rendering context.

V-GVolume
An interface to Jitter jit.gl.volume for 3D volumetric objects. Requires an input video stream.

V-G3dWord
An interface to Jitter jit.gl.3dtext object for text/words in 3D. In the current version words must be save in a specific text file, which should be available in your max path as by V-Module installation (file “V-3dWord-words.txt” in (sub)directory “V-Module Max Presets”.

V-GMeshX

A useful and versatile OpenGL Plotter, which accepts anything is an OpenGL 3D vertex matrix and allows advanced OpenGL functionalities. It includes all the features of the V-GMesh, plus the following new features:

- Blending:
  - blend_1 to set the blending mode for source mode
  - blend_2 to set the blending mode for destination mode
- Cull-facing (cull_face parameter)
- Capture to texture (see "cpt" menu and destination texture name)
- Input Texture and texture plane settings (see "tx" menu for source texture name and parameters under the section "T/S planes")
- Multi bus for efficient 3D object cloning (see "nm" menu for selecting multi bus name)

V-GNurbX

An interface to Jitter jit.gl.nurbs object. This allows rendering of Non-Uniform Rational B-Spline (NURBS) surfaces. Basically all the same in/out functionalities available in V-GMeshX are supported in V-GNurbX.

V-GMulti

A special V-OpenGLPlotter device which is able to pickup a multi bus (name) and operate on it. The V-GMulti can create from this a potentially very large number of clones of the original object, in hardware and efficiently. This device does not need a gstream, as this information is part of the multi name pointer.

It requires the following input matrices though:

- A position matrix, a gstream square 3 plane matrix with the coordinates of the center position for each clone. The number of clones is the result of the matrix dimensions (i.e. a 4x4 dimension result in 16 clones). This matrix input is mandatory.
- A rotation matrix, a gstream square 3 plane matrix with the rotation angles on x, y and z for each clone. This matrix input is mandatory.
- A scale matrix, a gstream square 3 plane matrix with the scale coefficient on x, y and z for each clone. This matrix input is mandatory.
- A texture matrix, a gstream square 3 plane matrix specifying the textures to be applied to each clone (each one referenced numerically). This matrix input is NOT mandatory, if the "text_off" toggle is in use.
- A color matrix, a gstream square 3 or 4 plane matrix specifying the color to be applied to each clone. This matrix input is NOT mandatory, if the "col_off" toggle is in use.

For all other parameters use the V-GMeshX reference.

V-GTexture

The V-GTexture device is useful for working on OpenGL textures. It can take a video stream as input (videoXX or streamXX) and create a pointer to a texture name (video-to-texture) or, vice versa, take a texture (capture from another device) and send it out as a video stream for further processing (texture–to-video).

Parameters:

- "tx/nm" = this specifies the texture name, both in case of video-to-texture and texture–to-video
- str_in = specifies the input video stream bus for video-to-texture applications
- flip = swap the video input, as OpenGL has a different orientation than Jitter does
- q-rate = output frame rate
- format = size of generated video stream if texture-to-video is applied
- cp_out = specifies the output (capture) video stream bus for texture-to-video applications
- rend_to = specifies the name of the render context the texture must be part of.

Note: a single V-GTexture device can work either as a video-to-texture or as texture–to-video device, but not both. In case you need to combine such functionalities, use two separate instances.

V-GSketchListY and V-GSketchListK
Wrappers for complex 3D scene sketching (based on jit.gl.sketch). These devices (mainly differencing in the number of drawing layers they support) allow you to write complex 3D scenes as "scripts". These scripts are text files (must be saved as ".skt" files) and consist of a sequence OpenGL instructions, expressed using the same coding of ji.gl.sketch. Moreover the scripts provide a few additional (proprietary but useful) extensions:

- Mapping of any script parameter to the Live GUI, expressed as a parametric modifier. A parameter modifier line must start with symbol "!". See next for more details.
- Support selective command activation/deactivation in order to dynamically expand/reduce a given scene (command-list) through Live's GUI. This kind of lines start with symbol "@". See next for more details.
- Support for comment lines in the script (for readability). These lines must start with symbol "//". Credit: this device is partially inspired by Darwin Grosse "Sketchpad". See: http://cycling74.com/2009/08/31/creating-a-sketchpad-for-jitglsketch/

Hereby some details about the syntax usage in these ".skt" scripts.

Generic OpenGL API commands (in jit.gl.sketch form)

Basically (almost) any C-like OpenGL API instruction as given in the "Redbook" can be used. They need to be converted in jit.gl.sketch form by:

- - Having everything set in lowercase, loose the "()" and, if applicable, append parameters inline to the command. Example: glColor(0.1,0.2,0.3,0.4) becomes glcolor 0.1 0.2 0.3 0.4 in the jit.gl.sketch form.
- - Any OpenGL built-in symbolic constant, in addition to being converted to lowercase, need to loose the "GL_" prefix. Example: GL_CLIP_PLANE1 becomes clip_plane1 in jit.gl.sketch form.

It is not in the scope of this document to review the OpenGL API (which is by itself \\\ huge ///), so, if you are interested, please check the following links:
http://www.cycling74.com/docs/max5/refpages/jit-ref/jit.gl.sketch.html
http://www.opengl.org/sdk/docs/man/

Extension: Parametric Modifiers coupled to Live's GUI

The sketch script can have any internal parameter modulated by the Live's GUI of the V-GSketchList device, using the device numeric boxes labelled "sk_pN" (N = 1...8), for coarse modification, and "fineN" (N = 1…8), for fine modification. This mapping is done in the script by using a "!" line. This type of statement is interpreted as a replace statement for the previous line and maps a given command parameter using the $N syntax (N = 1…8, one-to-one mapped to the just mentioned device Live's GUI numeric boxes). These lines must FOLLOW the command which initialize the script parameter value.

Example:
glrotate 0. -0.866 -.5 0.
! glrotate $3 -0.866 -.5 0.

This combination is interpreted as follows:

- The command glrotate is applied and has initial values: (0. -0.866 -.5 0.)
- The same glrotate command is modified in its 1st parameter by the device 3rd numeric box (actually both the sk_p3 and fine3, in coarse/fine fashion), as expressed by the modifier line, starting with "!" and having $3 in place of the first glrotate parameter.

Note: In the V-GSketchList there is even a mechanism to extend the number of parameters beyond the 8 numeric coarse/fine boxes in the device GUI. An input matrix (“in” menu in the device) can be used to modulate any number of parameters. The matrix cell, indexed by scanning each row sequentially from left to right, maps to a $N symbol in the parametric modifier.

For example if a 20x20 matrix is used to modulate the script, a given cell (r,c) = (3 15), which is the cell in the 4th row and 16th column, maps to symbol $((r)*20 + (c+1)) = $76 in the script. PS: It sounds complicated, but it actually isn’t. Experiment and you’ll see.

Activators / Deactivators

The sketch script can be made modular by having parts / commands enabled/disabled interactively through the Live’s GUI using the enableN (N = 1…8) numeric boxes in the V-GSketchList device. This feature is GPU friendly and removes any processing of disabled commands (deleted in the command line passed to the GPU).

To apply an activator/deactivator for any script command, the same command must be selected. To do this a line starting with the @ symbol must follow the command. The @ symbol must be followed by the number N referring to the enableN (N = 1…8) numeric box in the device.

So, if in the script a "@ <number>" command is found, this is interpreted as a conditional enabler of the command it FOLLOWS. The <number> specifies which of the device N toggles (enableN, N = 1…8) is used as condition.

Example:
sphere 0.1
@ 3

This is interpreted as the command "sphere 0.1" to be active (enabled) if enable3 == 1, and disabled (removed) if enable3 == 0.

Combined Parametric Modifiers + Activators / Deactivators

The modifier and activator features can be combined together. To do this:
- Use FIRST the activator constructor ("@ <number>") as previously specified
- THEN add an additional line combining the “@” construct with the “!” construct.

Example:
glrotate 0. -0.866 -.5 0.
@ 3
@ 3 ! glrotate $3 -0.866 -.5 0.

The first line is the command (and initial value). The second line is the activator (If enable3 == 1, then the glrotate command is active). The third line combines this to a modifier (if the command is enabled AND there is an input on 3rd parameter in the Live GUI of the device, then the script command is modulated in its first parameter). Note: If enable3 == 0, then the glrotate command is deactivated ANDno modifier is applied.

Comments

Comments are just lines which are ignored by the interpreter. They must begin with the symbol "\\". It may be trivial to say, but comments are very handy, especially when you need to modify your own code after some time (reverse engineering is one of the worst plagues affecting coders …..).

A Script Example (copy to a text editor, save using “.skt” extension and load it into the device):

```
reset
// fixed nucleas
moveto 0. 0. 0.
glcolor 1. 1. 0. 1.
sphere 0.15
moveto 0. 0.07 0.07
glcolor 0. 1. 1. 1.
sphere 0.15
moveto 0.07 0.07 0.
glcolor 1. .8 .8
sphere 0.15
moveto 0. 0. 0.
//
// fixed electron paths
glcolor 1. 1. 1. 1.
shapeslice 100
shapeorient 90 0 0
torus 1. 0.01
glcolor 1. 1. 1. 1.
shapeslice 100
shapeorient 90 60 0
torus 1. 0.01
glcolor 1. 1. 1. 1.
shapeslice 100
shapeorient 90 120 0
torus 1. 0.01
//
// electron 1
glpushmatrix
@ 1
glcolor 1. 0. 0. 1.
@ 1
moveto 0. 0. 1.
@ 1
glrotate 0. 0. 1. 0.
@ 1
@ 1 ! glrotate $1 0. 1. 0.
sphere 0.1
@ 1
glpopmatrix
@ 1
// electron 2
glpushmatrix
@ 2
glcolor 0. 1. 0. 1.
@ 2
moveto 0. 0. 1.
@ 2
glrotate 0. -0.866 .5 0.
@ 2
@ 2 ! glrotate $2 -0.866 .5 0.
```

```
sphere 0.1
@ 2
glpopmatrix
@ 2
// electron 3
glpushmatrix
@ 3
glcolor 0. 0. 1. 1.
@ 3
moveto 0. 0. 1.
@ 3
glrotate 0. -0.866 -.5 0.
@ 3
@ 3 ! glrotate $3 -0.866 -.5 0.
sphere 0.1
@ 3
glpopmatrix
@ 3
//
glflush
```

3D OpenGL-Processors V-Module

The following device types fall in this category:

- "V-GShadingEffects" provides 3D surface shading (taxellation). The shaders (loaded on devices such as V-GPreShader.amxd) perform custom lighting, color, vertex manipulation, fog etc.).
- "V-GMaterials" (alternative to the previous one) performs taxellation using Cycling74 new Max 6 material feature.
- "V-GPostProcess-IO" does not do 3D taxellation but, instead, allows post processing of the 3D stuff. It requires first the 3D object/scene to be rendered to a texture (see "capture" option on 3D plotters and V-Mixer), as taken by the V-GPostProcessIN.amxd device, it then allows you to do normal video effect processing (i.e. Using the V-FX-GL-MultiEffect.amxd) and finnaly posts the results in 3D it then uses a tranparent video plane on top of the 3D scene (V-GPostProcessOUT.amxd).

CV control devices in V-Module

As explained in the introduction V-Control devices create and/or operate on CVsends for control purposes. They are comparable to your synth's LFO generators (or CV input of a modular analogue synth in the old days) and include actions such as "produce MIDI to/from CVsend", "modulate a Live parameter" and even more complex manipulations. These device are aware of Max for Live API and allow you to mix and connect different dimensions together.
V-CVLiveMod
This device enables LFO-like modulation of any Live device using the V-Module CVsend busses. In older versions of V-Module (prior to 0.5) modulation was achieved built-in to the destination device. This was actually restrictive and caused the Live undo-list to be filled in if modulation was active.

Since version 0.5 the modulation approach has changed: having the V-CVLiveMod gives V-Module a flexible tool for CV-to-parameter modulation for ANY Live device (including any music instruments or effect). Parameters:

- "CV_in" = Input CVsend
- static = set a static value to the destination (useful for testing in absence of a CV_in)
- gain = apply a gain to the input CV_send
- offset = apply an offset to the input CV_send
- device_id / remote device menu = select Live device to be modulated (param to be selected)
- param_id / remote parameter menu = selectparameter of the Live device to be modulated
- refresh device list = re-sync with live device configuration (required if a change is done to Live track position or new devices are added).

V-Track2CV
A device which tracks the X/Y location of a color area in a video stream. The colour target area is defined by the "high color" / "low color" range. All colors matching this range are potential matches. The output X and Y can be sent to a pair of CVsend streams.

V-LFO2CV
A device which generates simple low frequency waves useful for CVsend applications. The period is based on Live tempo and to any beat resolution allowed by Live.

Controls:
- type-X/curve = select the LFO type
- type-X/period = a multiplier of the main period for the X curve.
- type-Y/curve = select the LFO type
- type-Y/period = a multiplier of the main period for the Y curve
- type-Z/curve = select the LFO type
- type-Z/period = a multiplier of the main period for the Z curve
- +/- = if ON makes the oscillation to be between 0 and 1. Otherwise it is between -1 and 1.
- period = time resolution in sync with Live tempo at the selected resolution
- p/p = stop start the LFO's
- q-rate = interval of CVsend sampling
- CV_X = CVsend used for output of curve X
- CV_Y = CVsend used for output of curve Y
- CV_Z = CVsend used for output of curve Z

V-CVOp
A simple operator for scalar mathematical operations on CVsend.

V-CVSlide
Perform a smoothing filter on the input CVsend, useful in case this is an audio generated input for filtering out instability or peaks.

V-MIDI2CV
Takes a MID CC value and translates it as a CVsend. Three input/output combinations available.

V-CV2MIDI
Takes a CVsend and translates it to a MIDI event (note, CC or other).

V-Joy2CV
Enables a standard HID device (GamePad or Joystick) to be used as CV controller for three parameters. If coupled to the V-CV2MIDI device turns a HID device in a MIDI controller.

V-DroidSens2CV
A terminal device for Matt Benatan's Max/MSP control app for android as available at http://mattbenatan.com/). Allows an android handset to communicate (via Wifi) to you Max/MSP/Jitter software and therefore to V-Module providing android sensor measurements as CV send feeds.

V-DroidCtrl2CV
A terminal device for Matt Benatan's Max/MSP control app for android as available at http://mattbenatan.com/). Allows an android handset to communicate (via Wifi) to you Max/MSP/Jitter software and therefore to V-Module providing a target for the control sliders and toggle enabled in the androids app. Combined with the V-CV2MIDI device turns an android device in a MIDI controller.

Audio devices in V-Module

As explained in the introduction, these devices can analyze a Live's audio feed, interpret it algorithmically and output, depending on the case, streams such as video, OpenGL, CVsends etc.

They allow you to do things such as audio modulating video, audio modulating MIDI, audio generating/modulating 3D coordinates, etc. As they require audio from Live they cannot be MIDI effects. In fact in V-Module they are all audio effects.

Note: in my opinion the best way to use a V-AuScope is to use them in a return track. This allows you to do bus feeding, to mute the track if needed and to keep the other tracks clean in terms of devices.

V-Au-Beat
An audio effect which can identify and auto-set Live tempo based on an audio feed. Note: it requires Oliver Pasquets's "op.beatitude" external to be installed (http://www.opasquet.fr/op-beatitude)

V-Au-BeatCVEnvelope.amxd
An audio effect which can identify and auro-set Live tempo based on an audio feed and generation of LFO CVsend envelopes. Note: it requires Oliver Pasquets's "op.beatitude" external to be installed (http://www.opasquet.fr/op-beatitude)

V-Au-Beat
An audio effect which can identify and auto-set Live tempo based on an audio feed. Note: it requires Oliver Pasquets's "op.beatitude" external to be installed (http://www.opasquet.fr/op-beatitude)

V-Au-SpectralEnvMod.amxd
A device analizing the spectrum in the audio signal and producing modulation.

V-Au-ScopeGLMatrix.amxd
Analyzes the envelope of a signal and produces a time shifting gstream (vertex matrix) which can be used and process uing V-Module OpenGL processors and generators.

V-Au-SoundShapeGLMatrix.amxd
Analyzes the envelope of a signal and produces a time shifting gstream (vertex matrix) which can be used and process uing V-Module OpenGL processors and generators.

V-Au-SpectralGLMatrix.amxd
Analyzes the spectrum of a signal and produces a gstream (vertex matrix) containing the amplitude values of each band. It can be used and process uing V-Module OpenGL processors and generators.

Extending V-Module (interfacing)

Here some specifications on how you can build new devices which can communicate with other V-Module devices. For each type of stream previously mentioned the following tables specify the interface to use.

"**stream**" busses

Stream name in the menu	**Send/Receive object name (must match)**	**Notes**
stream1	voutput1	**Send/Receive options** the send/receive names must match, as they must be resolved by Max. The names you put in the menus are not mandatory, but you are recommended in using the same the author originally did, for readability. Do not use the "---" option, as the send/receive must potentially scope multiple devices. You may wish to put a live menu selector to specify the active selection and use a "switch" for input selection and a "gate" for output selection. **Data Type** Keep in mind that the type of data used by video stream busses should be a jit.matrix of 4 planes (RGBA) of any type (from char to float32). As a reference open one of the V-Effect devices (having both input and output video streams). **Input** For instance if you open the V-FX-Slab device on the top portion of it you will see a "switch 25" object. The "r videoxxxx" objects are the input video stream busses you need to use. In case of input you DO NOT need to provide clocking (metro/qmetro). **Output** If you open the V-FX-Slab device on the bottom portion of it you will see a "gate 25" object. The "s videoxxxx" objects are the output video stream busses you need to use. In case of output you MAY NEED to have clocking (metro/qmetro) as part of the logic of your patcher (i.e. if you generate matrixes / video content).
….	….	Same as above
stream16	voutput16	Same as above
video1	video1	Same as above. The naming of this bus is actually been chosen as meant for final chain output. Anyway there is no difference in use from other video stream busses.
….	….	Same as above
video8	video8	Same as above

"**gstream**" busses

Stream name	Send/Receive name	Notes
gstream1	glstream1	**Send/Receive options** the send/receive names must match, as they must be resolved by Max. The names you put in the menus are not mandatory, but you are recommended in using the same the author originally did, for readability. Do not use the "---" option, as the send/receive must potentially scope multiple devices. You may wish to put a live menu selector to specify the active selection and use a "switch" for input selection and a "gate" for output selection. **Data Type** Keep in mind that the type of data used by video stream busses should be a SQUARE jit.matrix of 3 planes (XYZ) possibly using float32 values. See http://www.cycling74.com/docs/max5/tutorials/jit-tut/jitterappendixb.html for more specifications. As a reference open the V-GLOp device (having both input and output OpenGL matrix streams). **Input** On the top portion of the V-GLOp device you will see a "switch 17" object. The "r glstreamXX" objects are the input OpenGL stream busses you need to use. In case of input you DO NOT need to provide clocking (metro/qmetro). **Output** On the bottom portion of the V-GLOp device you will see a "gate 17" object. The "s glstreamXX" objects are the output OpenGL stream busses you need to use. In case of output you MAY NEED to have clocking (metro/qmetro) as part of the logic of your patcher (i.e. if you generate OpenGL matrixes content).
….	….	Same as above
gstream16	glstream16	Same as above

"**render**" names

Render name	Send/Receive name	Notes
mrender1	Not applicable	**Naming is crucial** A render context is a *named location* for Max and the GPU. It is (or directly refers to) a "pointer" to a memory location. Therefore you do not send/receive data in this case, but just refer to the place

		you are going to get data from and/or send data to. **Output render names in a menu** The trick I used in V-Module for selecting an output render is use the “drawto $1” message for all jitter GL objects wishing to render something. The NAME you use as input to the message MUST match the name used by the jit.gl.render object built in the V-OpenGLRender device you are targeting. As a reference open the V-Mesh device, and look for the “jit.gl.mesh” object. You will see that there is no parameter in the object itself, but an entire cloud of patches sending messages to it. The one you are looking for is the message “drawto $1”. The menu above it is used to select the name of the rendering context you like to use for out put. You MUST use these names if you want to render the output of a device in the OpenGL rendering context of a V-Module device such as the V-Mixer (or type V-OpenGLRenderer). **Input render names hardcoded in render devices** As the jit.gl.render must have a named location in its call, these names are hard coded all the patchers of type V-OpenGLRender and in the V-Mixer. In order to allow multiple render destination, each rendering device uses a different location name. Therefore I normally used a number at the end of the name. If you see a “1” at in the device name of a rendering device, then the device normally uses the **default render name “mrender1”**. If there is another number (for the devices in the “extra instances” maps, the number N should match the render name “mrenderN”. If you are not sure, just open the device and look for the jit.gl.render object. **Note about V-Effect devices type GL** V-Effect devices type GL require their own unique render context name for internal processing (offloading to GPU). These are not rendering contexts you should use as output of your devices.
mrender2	Not applicable	All explained above applies. This is a secondary rendering context name.
mrender3	Not applicable	All explained above applies. This is a secondary rendering context name.
mrender4	Not applicable	All explained above applies. This is a secondary rendering context name.
mtext1	Not applicable	All explained above applies. This is a rendering context name only used by the V-GRenderText1 device.
mtext2	Not applicable	All explained above applies. This is a rendering context name only

		used by the V-GRenderText2 device.
mtext3	Not applicable	All explained above applies. This is a rendering context name only used by the V-GRenderText3 device.
mtext4	Not applicable	All explained above applies. This is a rendering context name only used by the V-GRenderText4 device.
mdraw1	Not applicable	All explained above applies. This is a legacy rendering context name.
mdraw2	Not applicable	All explained above applies. This is a legacy rendering context name.
mdraw3	Not applicable	All explained above applies. This is a legacy rendering context name.
mdraw4	Not applicable	All explained above applies. This is a legacy rendering context name.
mdtext1	Not applicable	All explained above applies. This is a legacy rendering context name.
mdtext2	Not applicable	All explained above applies. This is a legacy rendering context name.
mdtext3	Not applicable	All explained above applies. This is a legacy rendering context name.
mdtext4	Not applicable	All explained above applies. This is a legacy rendering context name.

"**CVsend**" busses

Stream name	**Send/Receive name**	**Notes**
CVsend1	MIDIcc1	**Send/Receive options** the send/receive names must match, as they must be resolved by Max. The names you put in the menus are not mandatory, but you are recommended in using the same the author originally did, for readability. Do not use the "---" option, as the send/receive must potentially scope multiple devices. You may wish to put a live menu selector to specify the active selection and use a "switch" for input selection and a "gate" for output selection. **Data Type** Originally a Cvsend was thought as an integer between 0 and 127 (for MIDI direct compatibility), anyway nothing prevents you from using different scalar types, as long as you keep in mind to be compatible with the existing V-Module device (all supposing the number they get is an integer between 0 and 127). **Input**

		As a reference open the V-CVOp device and look for three “switch 17” objects on the top part of the patcher. The “MIDIccXX” objects are the one defined as input CVsend's (the name let's you see I originally thought of them as a way of getting MIDI control for parameters, only lately evolved as the LFO-like concept as it is now). **Output** As a reference open the V-CVOp device and look for “gate 17” object on the bottom part of the patcher. The “MIDIccXX” objects are the one defined as output CVsend's. **Applying a Cvsend** A CV send is meant to modulate a parameter you can also control manually. So if you open a patcher with a so called LFO-BANK you will see that the CVsend is sent as “alternative” input to the parameter I wish to modulate. At this stage the names of the target modulate parameters offered by the user interface menu are just p1..p32 (which makes it a bit tricky).
….	MIDIcc15	Same as above
CVsend16	MIDIcc16	Same as above

"**texture**" busses (as sources to a OpenGL object / V-OpenGLPlotter)

Render name	**Send/Receive name**	**Notes**
texture1	Not applicable	**Naming is crucial** texture names (as sources to an OpenGL object / V-OpenGLPlotter) are pointers which can be used to couple a source image or video stream to a GL *texture name* for texturing the 3D object (i.e. as the “skin” of the object). Any V-OpenGLPlotter is then able to pick this up as a texturing source. **V-GTexture as a flexible texture name mapper** In V-Module some older V-OpenGLPlotters could take a video stream directly as a source for an internal texture. This method, though, was not flexible enough for all texture applications and since version 0.6 has been generalized by creating a new V-GTexture device. This device can take a video stream as input (videoXX or streamXX) and creates a pointer (texture name). Any V-OpenGLPlotters is able to pick this up (as part of their common render context) by using a menu (generally identified by the description “tx” in the device M4L presentation GUI). In the code bounding is done using a string (symbol) appended to the “name” command of the jit.gl.texture object. Currently four separate texture names are available by default (texture1 to texture 4) in the menus.

		Note about V-Effect devices type GL V-Effect devices type GL require their own unique render context name for internal processing (offloading to GPU). These are not rendering contexts you should use as output of your devices.
...	Not applicable	All explained above applies. This is an additional texture name as possible source.
texture16	Not applicable	All explained above applies. This is an additional texture name as possible source.

"**texture**" busses (as rendering destination for a OpenGL object / V-OpenGLPlotter)

Render name	**Send/Receive name**	**Notes**
texture1	Not applicable	**Naming is crucial** Besides being used as sources for texturing, textures can be used as a special render destination, useful for (real-time) post-processing of a 3D object or entire scene. Again, texture in this case are no send or receive, but *names*. **Support by V-OpenGLPlotter as rendering "captures"** Many V-Module V-OpenGLPlotter devices offer now an alternative rendering destination in the form of texture *captures*. These can be selected using an additional menu (identified in the M4L presentation GUI by the "cpt" description) which selects a texture name to render to (the reder context must also be specified, as usual). This texture can then be picked up, as a source texture somewhere else (by another V-OpenGLPlotter for instance) or used by other devices which enable post-processing as video stream (i.e. the V-GTexture). In case post processing is done using a V-Effect type open GL the readback (backread) technique is not needed. **V-GTexture as a flexible texture name mapper** The same V-GTexture device previously mentioned, beside mapping a video stream to a texture, can work the other way around: take a texture (capture) name and send the data out as a video stream for (2D) post processing. **Post processing V-OpenGLPlotter devices** Some new developed OpenGL plugin can take a texture name and do some 3D-like effect on the texture directly in OpenGL (using shaders). This is the case of the V-GTexGlow plugin. **Feedback internal to the V-Mixer** Even if not explicitly mention as texture, the feedback mode in the V-Mixer new incarnation ("V-GL-RenderMixer4RGB-Fdbck") is actually a rendering to texture technique. In this way the entire 3D scene can be captured as a video stream which can be used by the V-Mixer itself in one of its stream input!!! Very, very psychedelic…..

		Note about V-Effect devices type GL V-Effect devices type GL require their own unique render context name for internal processing (offloading to GPU). These are not rendering contexts you should use as output of your devices.
...	Not applicable	All explained above applies. This is an additional texture name as possible target destination.
texture16	Not applicable	All explained above applies. This is an additional texture name as possible target destination.

"**objects**" busses

Render name	**Send/Receive name**	**Notes**
object1	Not applicable	**Naming is crucial** **multi** busses are actually *names* which specify, within a render context, a destination place to hold the 3D characteristics of an original object for another, special object/device (V-GMulti, based on the jit.gl.multiple object in Jitter), to use as source information to use for creating clones. **As output by a V-OpenGLPlotter** The newest V-OpenGLPlotter devices in V-Module (such as V-GMeshX) allow you to specify a multi name. This makes its information available to a V-GMulti device to pick up for creating clones. **As input to the V-GMulti device** In V-Module a special V-OpenGLPlotter device called V-GLMulti which is able to pickup a multi bus and operate on it. The V-GMulti can create from it a potentially very large number of clones of the original object, in hardware and efficiently. This device does not need a gstream, as the this information is part of the multi name pointer.
….	Not applicable	All explained above applies. This is an additional multi name.
object16	Not applicable	All explained above applies. This is an additional multi name.

"**shaders**" names (for selecting a given shader

Shader name	**Send/Receive name**	**Notes**
shader1	Not applicable	**Naming is crucial** The name must match with the name assigned by a device loading the actual shading program (such as “V-GPreShader.amxd”).
….	Not applicable	All explained above applies. This is an additional name.
shader16	Not applicable	All explained above applies. This is an additional name.