Sumu

Spectral + FM synthesis with spatialization

Prerelease DRAFT – Early Access version

Madrona Labs

Version 1.0 (prerelease), May 2024. Written by George Cochrane.

Illustrated by David Chandler.

Typeset in Adobe Minion using the TEX document processing system.

Introduction

"Little by little you must create a fog around yourself ... until nothing is any longer for sure, or real."
–*Carlos Castaneda*

Sumu is a software instrument that combines additive synthesis, FM, and fine-grained spatialization in an easy-to-use semi-modular instrument.

Each Sumu voice is made up of 64 independent pairs of oscillators with FM and noise. Every pair has its own lo-passable gate and its own motile position in sonic space. This gives you lots of sonic potential, but it may sound like a lot to wrangle.

"I guess that stuff's just for tweed-jacketed CS majors, or film composers with DSP-powered magic boxes," you might say. You swirl some roiboos dregs in an earthenware mug. It seems time is always moving on. There are arrows to loose and ears to feed, and now is not the moment.

For more on the theory behind Sumu, see Chapter 1, *"Parallel Pathways in Perpetuity"*

Luckily, Sumu gives you three easy ways to put all of this power to work.

- *PARTIALS* module - Turns interpreted audio into a map of pitch, level, and noise data over time, for each partial to follow.
- *ENVELOPES* module - One ADSR envelope per partial, with individual triggering and global shaping.
- *PULSES* module - One probability gradient controlling two shape-blasters for each partial, with strong individuality.

This makes for an instrument that lets you scratch your additive itch to great effect, without getting lost in the fog.

A Quick Overview

Sumu's interface has four main sections, from top to bottom:

- The *header* section, where you'll find the patch menu, along with the version and registration status of your copy of Sumu.
- The *shapes* section, where control data (and more) flows from the *INPUT*, *PARTIALS*, *ENVELOPES*, and *PULSES* modules.
- The *patcher* section, where connections between modules are made and broken.
- The *audio* section, where signals are viewed (*SCOPE*) and sound is created (*OSCILLATORS*), controlled (*GATES*), placed in *SPACE*, and colored (*RES*).

However, there are always further secrets enshrouded in the mist. Let's take a stroll through the forest.

Fog Farmer's Almanac

If Sumu is your first foray into anything modular, or your first bite of the additive apple, let this manual be your guide. Whatever your perspective, we mostly hope that you have fun exploring this new instrument, making sounds you've never heard before.

This manual is arranged in five sections:

- Parallel Pathways in Perpetuity Get a little background on the concepts behind Sumu.
- Getting to Know Sumu Find out how Sumu fits into your system, and learn to use the various types of controls and connections that you'll find.
- Sumu: Module by Module Take a guided tour through the different parts of Sumu, module by module.
- Vutu: Your Audio Cartographer Explore turning audio into partials maps for use in Sumu, using the included desktop app.

1 Parallel Pathways in Perpetuity

Let's talk a little about *Additive Synthesis*. What's it all about? Given how uncommon it is, it's probably easier to think about how it diverges from more frequently used forms.

- Subtractive Synthesis: Create an unceasing, unholy racket, then pare it down to something useful by changing volume and tone over time.
- Frequency Modulation Synthesis: Bash tightly controlled waveforms into one another until they agree to form the sound you want.
- Additive Synthesis: Build your perfect world of sound, molecule by molecule, millisecond by millisecond.

Stated that way, additive sounds pretty genteel in comparison, *non*? But maybe a bit of time-waster, if you don't tend toward micromanagement. And what's this "molecule" business, anyway?

Well, you may have heard it said that if you look close enough, *every* sound is made of plain old sine waves. Even the noisiest scronching wail looks like countless little interacting wiggles under the microscope, because physical objects, air, and even oscillators are (in a word) elastic. Nothing goes from 0 to 100 in an instant; there is always a swing.

"I say, I say, I do say... Where there's a swing, there's a sine."
–Foghorn Synthorn, Esq.

So the story goes, if we know (or can guess) the frequency makeup of a given sound over time, we can simply rebuild it using lots of sine wave oscillators tuned across the frequency spectrum, controlling their amplitudes with sophisticated envelopes.

Early additive synth systems dared you to do just that; exhaustively plot out volume relationships for dozens (or hundreds) of partials over time. Get it just right, and you could have a startling true-to-life flugelhorn under your fingers, free of sampler-esque pitch and time artifacts.

Get it wrong (or simply give up after realizing you like to make music, not spreadsheets) and it's time to pull up someone else's preset again. What a bore.

Later entrants into additive synthesis dreamed up lots of clever ways to wrangle partials en masse—everything from big touch-sensitive LED grids, to interpreting images as spectrograms, even analyzing and resynthesizing audio.

Sumu focuses on the latter strategy, letting you creatively process and analyze your own audio using the Vutu desktop app, then bring the resulting analyses into the ***PARTIALS*** module to control the multitudinous sound-atoms inside.

And if that's all it did, that'd be fine. Not unique maybe, but useful. Where Sumu diverges from other such additive instruments is in its modularity, and the peculiar and well-situated array of modules on offer.

You might think, *"Modular? My goshdarned shoes have patch points these days. What's so special about that?"* Well...

Incidentally, if your shoes DO have patch points, please send us your cobbler's number.

The Widest Highway

When you patch between many of the modules in Sumu, that cord doesn't carry just one signal, or even one signal per active voice. It instead carries *64*, one for each partial. That also means that many of the modules actually act as 64 independent modules, one for each partial across the spectrum. This means 64 envelopes, pulse generators, oscillator pairs, gates, even a 64-channel oscilloscope.

Hit one of those mega-modules with a simple signal (like the ***pitch*** or ***gate*** output from *INPUT*), and all 64 channels move as one. Connect a complex signal instead (such as the ***pitch*** out from *PARTIALS*), and suddenly, things go kaleidoscopic. Every channel of the module reacts individually to the control data for its own partial.

Fire all the envelopes. Tune all the oscillators.

To put that in perspective, that means you could:

1. Load up your favorite rooster's best crow as a partials map in *PARTIALS*
2. Patch its ***noise*** output into *SPACE*'s ***speed*** input. Now each oscillator pair shifts its place in space a little quicker when its corresponding playhead in *PARTIALS* passes a noisy spot in the cock-a-doodle-doo.
3. Patch its ***amp*** output into *PULSES*' ***bpm*** input, and the pulse outs into *GATES* (set to ***lo-pass***). Now, the two shape-generators for each partial run faster when that part of the sound gets louder, and the unique pulses from each one ripple through the timbre of its corresponding oscillator pair.

Think locally, patch multifariously.

Sound a little nuts? Well, it is. But only on electronic paper. Dive in, and you'll catch the scent quickly.

2 Getting to Know Sumu

This chapter shows you this instrument fits into your computer ecosystem, and gives you a working knowledge of the various types of controls you'll find throughout. Some of them act just like you would expect, but some have more personality than that.

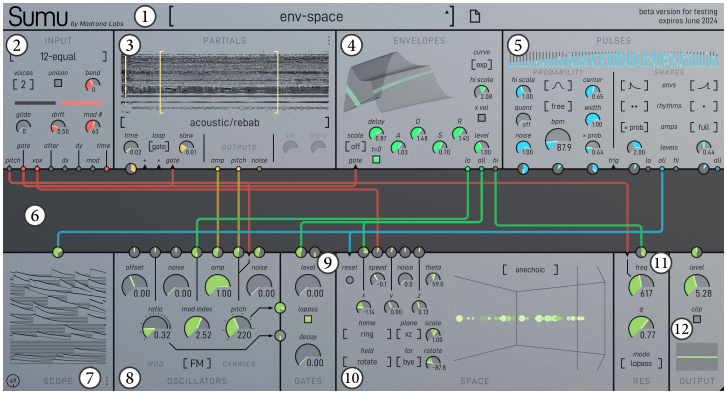
First Things First

Sumu is a software *plug-in* that comes in both VST and AU formats. A plug-in does not run on its own. It runs within an application (known as a *host* or *Digital Audio Workstation*). Some good hosts include Ableton Live (on Mac and Windows), Logic and Numerology (Mac), and FL Studio (Windows). All of these hosts provide easy ways to use plug-ins like Sumu.

Every host handles instruments a little differently, so for more information on using instrument plug-ins in your own system, please see the user guide that came with your DAW.

If you run into any trouble when installing or operating Sumu, please search our forums at ***http://madronalabs.com*** or, if that fails, send your questions to ***support@madronalabs.com***. We (and the growing community of Sumu users) are here to help you.

An Annotated Map of Sumu



This section gives a quick description of each of Sumu's areas, to give you a little familiarity before we go deep. For in-depth information on each module, control, and feature, see Chapter 3, *"Sumu: Module by Module."*

1. Header

   This area's most vital item is a nice big preset selection menu with back and forward buttons for fast switching. On the right, you can find your license information as well as the plug-in format for the current instance of Sumu. Click the "..." icon to access global settings, which set the way Sumu displays info and responds to MIDI & OSC.

2. *INPUT*

   This module receives note and control signals you send Sumu over MIDI or OSC, and makes them available to the rest of Sumu's devices.

If you have experience with CV-controlled synths, you can think of the Key module like a MIDI-CV converter box that outputs digital gates and ”control voltages.”

3. *PARTIALS*

   This module is one of the labor-saving tools Sumu offers. It takes audio (analyzed by the included Vutu app) and interprets it as a map of pitch, amplitude, and noisiness over time, for all 64 partials to follow.

4. *ENVELOPES*

   This module creates discrete time-based control vectors for each partial, commonly used to affect volume or timbre as notes are struck and/or held. While these envelopes are a fairly standard ADSR (Attack-Decay-Sustain-Release) type, there are tricky ways to shape them as a group, and a 64-channel trigger input that lets you get creative about firing them off.

5. *PULSES*

   This module provides two pulse generators per partial which always trigger in sync (each with its own timbral, rhythmic, and dynamic options). Trigger rate, probability, and density can differ for every partial, based on control inputs and a probability generator which applies shaped gradients of likelihood across the spectrum. This lets you make modulations that can be focused on a certain frequency range. You could also, for example, use the pulses to trigger all of the *ENVELOPES* at different times.

6. *PATCHER*

   The Patcher is the dark central strip in the plug-in window, surrounded on all sides by the Modules. The patcher lets you connect

signals from the outputs of modules to the inputs of modules. It is notable that multiple inputs can be fed from a single output, or multiple outputs to a single input. We think this is way more powerful and easy to use than a ton of menus.

7. *SCOPE*

   With multi-channel signals spurting forth from every modulator output, it's easy to lose touch with what's going on. This module shows you an evolving graph of every channel from the output you patch in.

8. *OSCILLATORS*

   This module hosts Sumu's oscillator bank (believe it or not). One carrier-modulator pair (plus noise) form the audible heart of each partial. All the juicy parameters thereof can be controlled individually per partial, under the influence of beefy 64-channel control signals from the modules above.

9. *GATES*

   If you're familiar with modular synth jargon, this module can be described as 64 VCAs (Voltage Controlled Amplifier/Attenuators) or LPGs (Low Pass Gates), depending on the mode you choose. Either way, *GATES* works in concert with signal sources like *ENVELOPES* and *PULSES* to change the level (VCA mode) or the level and timbre (LPG mode) of the input signals.

10. *SPACE*

    This module gives each of our 64 partials its own place across the sound field; both a home spot it tends to hang near, and a certain

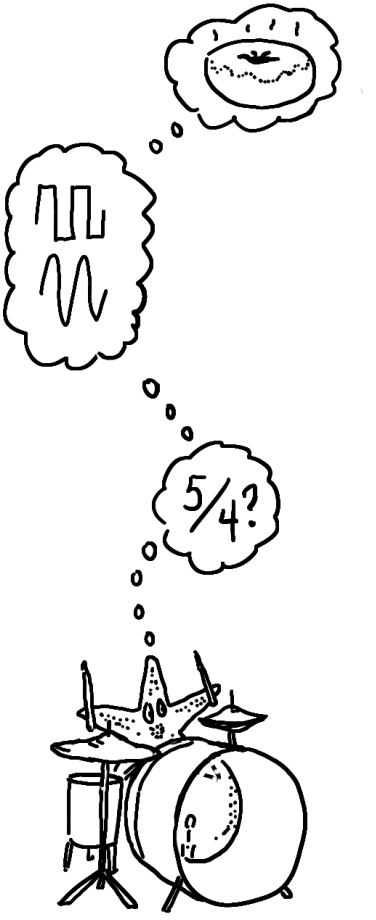
tendency to meander, according to the vector field you select. Speed and accuracy of movement are variable on a per-partial basis.

11. *RES*

This module is inspired by the Resonators section of the Polymoog synthesizer. It's simply a state-variable filter, with limited bandwidth and a bit of distortion to give it a little glow. In Sumu, a synth we could otherwise describe as "very digital," it's nice to have a built-in way to add a different flavor.

12. *OUTPUT*

The *OUTPUT* module is where all of your audio comes together before venturing out into the wilds of your DAW. It gives you control over the final output level, with a handy limiter to rein in roaring moments. It also contains a handy oscilloscope, which gives you visual feedback about the sounds you're making.



Presets

We labored heavily to make Sumu simple and inviting to use, but flipping through preset sounds is a good way to hear what it can do, and see how it's done. Sumu has room for both *user* and *factory* presets. The user area is where you'll keep your own creations, and where we put contributions from other Sumu users that we include. The factory presets are meant to be a small and well-rounded set of sounds that you'll come back to often.

Types of Presets

- *keys*

  The bread and butter of patches. Or the tofu on rice, if you prefer. When you want to start making a tonal song, head to this section and you're guaranteed to find patches that play notes in tune for you. Electronic and acoustic pianos are here; strings and things too.

- *pads*

  Like keys in that they play in tune, but are generally a bit more drawn-out. When playing a pad, you'll need to hold down the keys for a while for the sound to develop, and then it'll usually go someplace interesting.

- *drones*

  Like pads but farther out, and not always tonal.

- *sequences*

  These are sounds that could become a whole song; explorations, often with the pulses module, sometimes rhythmic. You know, the kind of thing that often comes out of a modular synthesizer. But in Sumu.

- *machines*

  Like sequences, but these presets may start making sounds whether or not you have a note held down. To avoid this being super confusing for the uninitiated, all presets like this are in this category.

- *environments*

Here we are, deep in Sumu-land. Hold a note and just wait... Partials may come and go. The experience of space is strong here. Flocks of weird creatures, synthetic streams or space station rooms. Is filmic ambiance your goal? You're in the right place.

- *percussion*

  Not what Sumu is known for, but it has some good percussive energy when it wants to. Cowbells, filter thwips, floppy bass drums and anything you could make a beat with go here.

- *techniques*

  Instructional patches that teach you how to do one thing or another, with a few clearly intentional patch cords.

- *vocals*

  Sumu does cool stuff with vocals, so we made a whole category to suit. Hear and play with our patches for fun, or drop in your own vocals in place of the partials sources here to start lofting your own productions into space.

Using Dials

So, you'd like to go beyond the presets? Of course you would! Meet *dials*. They're found in every module. Like knobs on any piece of gear, dials are mainly good for two things: manipulating signals and giving you information. However, whereas most knobs inform you merely about a single unchanging value they've been adjusted to, Sumu's dials act as tiny signal viewers as well. This means they not only show you the value

you've adjusted them to, they also show you the values they're being pushed and pulled to by incoming modulation signals.

To modulate a dial's signal, just make a connection to the dial's signal input in the patcher. Every signal that can be modulated has a signal input next to it—this is how Sumu can provide so much control without using menus. Signal inputs are like small dials without displays, or regular knobs, if you like. We'll cover the patcher and signal inputs thoroughly in a later section.

To set the position of a dial, you can do any of the following:

- Click in the dial's track (the dark area within it) to set the value to the click position. While still holding, drag up and down to adjust the value.
- Hover over a dial and use the scroll wheel to fine-adjust the position. At slow speeds, each click of the scroll wheel corresponds to the smallest currently visible increment of the dial. Scrolling faster accelerates the change.
- Click and drag vertically on a dial outside the track area to adjust the dial from the current position.
- Double-click or command-click a dial to return it to its default value.

Holding down the shift key before any of these motions are done will modify the motion to be a fine adjustment. This allows particular values to be set precisely.

Detents

Some dials, such as ***OSCILLATORS*** carrier pitch, have *detents*. Detents are useful default positions. For example, the pitch knob has a detent at an

A note in each octave (110Hz, 220 Hz, 440 Hz...) to keep the oscillator tuned to MIDI notes. Normal use of these dials makes them stop only on the detents. By shift-clicking a dial with detents, or holding down shift and dragging it, you can adjust it to any position in between the detents.

Numeric Displays

All of Sumu's dials show their current value both in the (often changing!) pointer position, and in a numeric display below each control. The numeric display does not show the modulated value, only the center value that you have set on the dial itself. The numeric displays are not directly editable, so just get that crazy idea out of your head.

We tried it the other way, and all those flashing numbers were a bit much.

The ***Show numbers*** toggle in the global settings on the header lets you turn off all the numerical displays, if you'd rather not see them.

Dial Scales

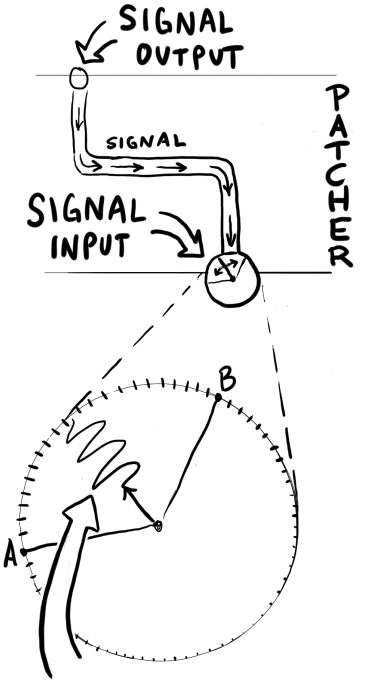
While many dials are linear (the change per degree from high to low is constant), some dials have logarithmic scales where the change is much larger as the value gets higher. This was done in cases in which a logarithmic scale matches the changes you perceive better than a linear one, as in oscillator pitch, for example.

In a logarithmic scale, equal movements of the mouse in different positions on the dial will produce differently-sized changes. For example, 55, 110, 220 and 440 Hertz are all equally spaced apart on the ***pitch*** dial in the ***OSCILLATORS*** module.

Using Buttons and Switches

There's not a lot to say here, only that switches need only be clicked to be toggled (you'll see the little dark switch move back and forth) and the same goes for the buttons. Dark is off, bright is on.

Using the Patcher



The patcher is the *grand connector.* With it, you'll bring together the tools Sumu offers, to do really fun stuff. The patcher is both the place from which much of the joy of working with Sumu springs, and the part of Sumu most likely to confuse you at first.

The Patcher is the large dark central area surrounded by all the modules. It lets you patch signals from the outputs of modules to the inputs of modules by drawing patch cords. Each patch cord has an arrow on it that shows which way the signal is flowing. Note that though signals tend to flow down, from *ENVELOPES* to *GATES*, for example, this isn't always the case, because inputs can be found on both the top and the bottom of the patcher.

There are no signals underneath the patcher that can flow up, but signals from above the patcher can go to other modules on top. And remember, modulation and audio signals are both the same thing, just made up of different frequencies, so it's perfectly fine to experiment by connecting any output to any input.

Some signals are *bipolar*, meaning they can have negative as well as positive values. Negative signals light up the outputs just like positive signals. In other words, the absolute value of the signal controls the output brightness.

Signal Outputs

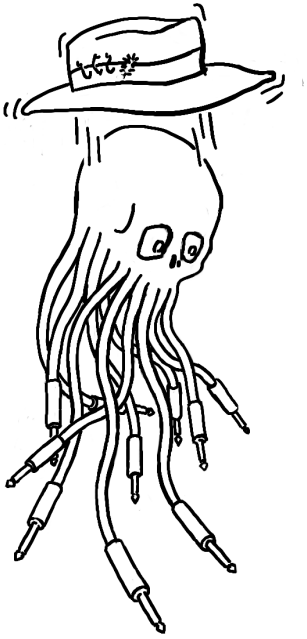
These are the tiny circles on the edge of the Patcher; the places from which all patch cords start. They light up to show the current value of the signal.

Signal Inputs and Modulation

These are the small dials bordering the Patcher; the places where all patch cords end. Each signal input connects to just one dial. When you connect a varying signal to an input, it modulates the dial's signal just as if you were moving the dial itself, but possibly at much faster rates.

Signal inputs are also knobs that let you adjust the amount of modulation applied to the dial. They do not display incoming signals themselves, because you can always see the effect in the dial. Some inputs are bipolar, meaning the value by which they multiply the signal can be either positive or negative. Like the bigger dials, each signal input dial has a default value, and returns to this value when double-clicked.

For example, the exponential pitch inputs to ***OSCILLATORS*** have a default value that corresponds to standard tuning when the pitch output from the ***INPUT*** module is connected. Changing this input dial makes nice music into weird tones very quickly. But by double-clicking to restore the default value, normalcy can be quickly restored if desired.



Envelope Trigger Input

Sumu's ***ENVELOPES*** module can accept multi-channel *trigger* signals, and it does so through a triangle-shaped input labeled "trig." You patch signals into this input just as described above. The big difference is that there is no dial to set the level of trigger input—a signal either triggers the intended response, or it doesn't. For this reason, you'll want to make sure that the signals you patch into the trigger input are appropriately "triggery," whether they're purpose-built trigger signals like the ***gate*** output of the ***INPUT*** module, or a particularly spiky control signal.

Patching

To make a patch cord, drag from an output to an input. As you drag, you'll see a glowing line with an arrow at the end stretch from one to the other. This shows the new connection you are making. This is a pretty nifty thing, since such routing does not involve deciphering menus or matrixes of things you can't see—you only need to look at what's on the screen, which is everything. This ease of use is intended to keep Sumu feeling like an instrument; something you can grab, pull, and mess with fluidly.

You can make patch connections while holding a note down, and they will affect the currently playing note just as patching a hardware modular would. By holding a note and touching a patch cord end to various signal outputs, you can even get intermittent glitchy sounds that are reminiscent of playing with a live electrical circuit, or *circuit bending*.

Multitudes Within Multitudes

Almost every module panel in Sumu's interface is really a controller for as many copies of the module as there are voices. And each voice has its own internal patcher. When a patch cord is made using the patcher UI, it is made simultaneously in the patcher within each voice. For example, if you connect an output from *PULSES* to *OSCILLATORS* pitch, you are connecting *PULSES* of voice 1 to *pitch* of voice 1, *PULSES* of voice 2 to *pitch* of voice 2, and so on. The *INPUT* module is the exception: it is more like one module with a signal output for each voice. When a note is played repeatedly, the *INPUT* module sends the note signal to each voice's patcher in turn to create polyphony.

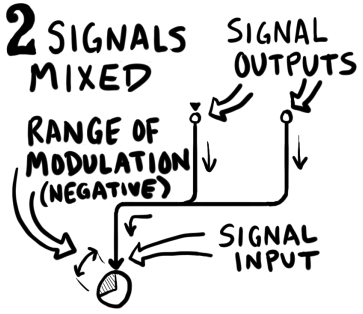
Since they are controlled by the common patcher UI, and one set of dials, the patch created for each voice is identical. But the signals that flow through each voice's patch can be very different. Thus, each voice is separately controllable, in timbre, modulation, and all of its parameters.

A patch cord always takes on the color of the module it is coming from. This helps you see at a glance what is going where. To modify a patch cord after it's made, first you must select it. To select a single cord, just click directly on it. When multiple cords are running over the point you click, clicking repeatedly will rotate through all the cords at that point. You can also select a group of cords in the patcher by clicking on an empty part of the patcher, then dragging over multiple cords.

Removing or Repatching a Cable

When a cord is selected, its *handles* are visible. Handles appear as circles at either end of the cord—you can drag them to move the ends. If multiple cords are selected starting or ending at the same place, clicking the handle there will move all of the selected cords.

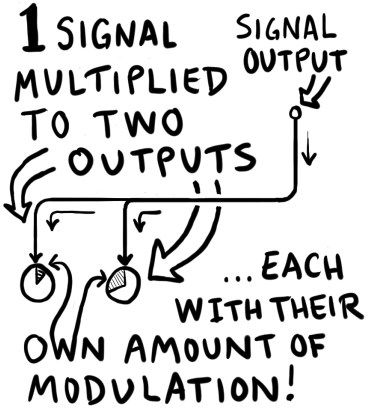
Typing the delete key should delete all the selected cords. You can also delete a patch cord by dragging either end to a place with no input or output. An X will appear instead of the handle at the end, and POOF. It's gone. Again, the changes happen in real time for any currently held notes.



Mixing and Multing

If multiple cables go to a single input, the signals are *mixed* together. The sum of all these signals is then multiplied by the input dial value. If multiple cables go from one output to more than one destination, the signal has been multiplied, or *multed.*

That's not terribly important information, but it's good to have your terminology straight, especially if you move on to other modular instruments.



Unipolar vs. Bipolar

Some output signals, such as the envelope outputs, send only positive values, and are *unipolar.* Others (like the *pitch* output on the *INPUT* module) swing both positive and negative. These are *bipolar.* Negative and positive signal values follow all of the same rules that real numbers do in algebra.

For example, if a negative-valued signal is multiplied by a negative signal input dial, its effect on the modulation will be positive.

Default Signal Routing

We've seen how easy it is to patch Sumu's modules together, but it's important to know that some connections in Sumu are pre-made for you.

The *OSCILLATORS* → *GATES* → *SPACE* → *RES* → *OUTPUT* signal path is pre-routed. Small dials between some of these modules (that suspiciously resemble the Signal Inputs we saw in the Patcher, if you ask me) act as signal level controls.

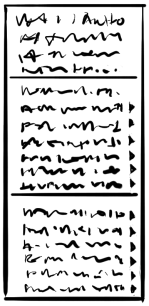
So there's our voice, pre-patched (to a degree!) Of course, the control and modulation modules on the top row can (nay, must) then be brought into play at your discretion. In this way, you might visualize the bottom row of modules as the troops, and the top row as their wildly-gesticulating corporal. You and your DAW of choice, of course, are the four-star generals in this analogy. Sorry, I think this granola might be really old.

3 Sumu: Module by Module

This chapter will take you on a more detailed tour of the various modules that compose Sumu, one by one.

The Header

Apart from reminding you which amazing plug-in you're currently using, Sumu's header mainly deals with patch access and management, and user interface options. All the things that don't affect the sound, in other words. The big drop-down menu in the middle displays the current patch name, and lets you select patches from a hierarchical list.



The patch menu

The drop-down patch menu has three main sections. The first section holds the Copy, Paste, and Save commands. When you select "Copy to clipboard," the current patch is saved in a text-only format that you can paste into other text documents.

This lets you send a patch to a friend in an email, or post it on a forum, for example. "Paste from clipboard" does the reverse.

"Save as version" lets you quickly save a variation of the current patch (with whatever tweaks you may have made since loading it) without having to enter a new patch name. The new patch is named after the current patch, followed by a revision number in brackets, incrementing with each new version you save.

"Save" permanently updates the current patch with any parameter changes you've made since loading the patch. This is, by definition, a bit risky, unless you're sure of the changes you've made. In many cases, you'll be safer using "Save as version" or "Save as..." when making incremental changes to an existing patch.

"Save as..." brings up a file chooser from which you can create a new file to save the patch to, or choose an existing one to overwrite. Patches from the Audio Units version of Sumu are saved in the .aupreset format. This is a compressed XML format, compatible with Logic, Live and other Audio Units-friendly applications. Patches from the VST version of Sumu are saved in the .mlpreset format. This is the same XML format, but uncompressed.

"Revert to saved" returns all parameters in the currently loaded patch to their original, saved settings. You can also activate the Revert to Saved feature by sending MIDI program change 128 to Sumu. This can be useful when recording multiple takes of Sumu dial-twiddling as audio in Ableton Live. In the Clip View for the MIDI clip you're working with, set the "Program" parameter to 128. Each time that MIDI clip is launched (with its launch button or a stop-and-start of the transport), Live sends program change 128 to Sumu, reverting the patch to its saved value. This gets you back to a consistent starting point for the next recording pass.

Below the commands are all of Sumu's patches in two more sections: the so-called "factory presets," in directories all starting with "Sumu,"

followed by storage space for your own personal patches. Some user presets, contributions from fellow Sumu users, are installed here by default.

Thanks, wonderful sound-crazed Sumu users, for all your contributions!

Presets are all stored on your hard disk in the right place for user data on your operating system of choice. For simplicity, factory presets are stored in the same place as your own patches. In the unlikely (but perfectly valid) scenario that you have multiple people with accounts on the same computer all using Sumu, each person can have a copy of the factory presets along with their own patches in their home directory.

On Mac OS, the user directory is in *(your home directory) / Library / Audio / Presets.* On Windows, it's in *(your home directory) / UserData.* If you're trying to copy your preset files using Windows Explorer, be aware that even though it's the recommended path for user data, Windows makes this directory invisible by default. Likewise, Mac OS 10.7 Lion and more recent versions hide the Library folder. You can navigate to the Library folder in the finder by choosing the "Go" menu and holding down the option key.

Selecting patches via MIDI

If you'd like to be able to load patches by sending MIDI program changes to Sumu, create a folder titled "MIDI Programs" (note the capitalization and the space between words) in one of the following locations, depending on your platform:

- Mac OS: /Library/Audio/Presets/Madrona Labs/Sumu
- Windows: (Your home directory)/UserData

Copy the patches you'd like to access with MIDI program changes into the "MIDI Programs" folder you've created. The folder is scanned by Sumu on startup, and the presets in it are assigned numbers, in alphabetical order. To rearrange the programs, give them new names so they are in a different alphabetical order. Send a program change to Sumu that corresponds to your chosen patch, and Sumu will dutifully switch to that patch.

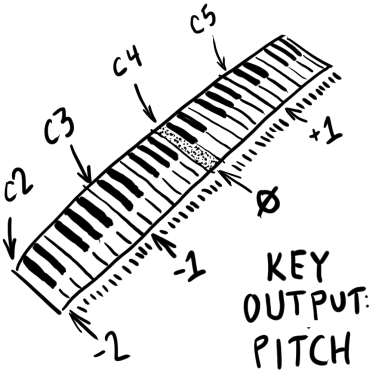
If you look at the MIDI Programs folder in Sumu's preset menu, you will see each preset listed, followed by its MIDI program change number.

Global settings

Click the "..." button on the right of the header to access the global settings. ***Show numbers*** lets you toggle the numeric displays that sit next to each dial on or off. ***Animate dials*** lets you toggle the animated scope-like displays inside the dials on or off. ***Reset editor size*** returns Sumu's window to its default size. The ***Input protocol*** menu lets you specify whether you wish to use MIDI or OSC to control Sumu. The ***OSC port offset*** menu lets you specify the OSC port you wish to use for control. For example, if your base OSC port is 6000, and you set the offset to 5, Sumu looks for OSC control signals on port 6005.

Version and registration

The right top corner shows the version of Sumu you are running, as well as your registration info. When you purchase a copy of Sumu for yourself, we encode your name and account information into it. This shows to you and the world that you are supporting what we do—from our end it means we have agreed to help you out with Sumu if any problems arise, and to maintain and improve it.



INPUT

This module receives all the MIDI data you send Sumu's way, and turns it into useful control signals that you can route to other modules with the Patcher. If your controller were a basket of fresh fruit, you could think of *INPUT* as a robot, blending juices the other modules thirst for.

Tuning menu

The menu up top selects the tuning table Sumu uses to map incoming MIDI notes to specific frequencies. *12-equal,* the default tuning, is short for 12-tone equal temperament. It is the basis for most modern Western music, but there are around a hundred others to try, included with Sumu. There are too many scales to describe here, but if you open up the .scl file for a scale you're interested in, you can read it as text, and often find a bit more information that can lead to an article on the subject. As a start, we've selected some of the public-domain scales from the Scala archive and sorted them into the tuning menu according to what musical culture they're from.

You can also add tuning files in .scl format to the scales directory yourself, or make your own using the free software Scala, available at *http://www.huygens-fokker.org/scala/.*

Voice controls

- *voices:* Sets the number of voices of available polyphony, from 1-16. The row of lights below illuminate and go dark one by one as voices turn on and off, showing you how many voices you have left to employ.

- *unison:* Toggles Unison mode on or off. Unison mode combines all available voices into one monophonic voice, which can make for some very big sounds. If multiple oscillators at exactly the same pitch are added together, the result can sound quieter than a single oscillator because the waveforms cancel each other out. This is hardly ever what anyone wants, so Sumu's sound engine applies a small random

frequency drift to the pitch of each oscillator to maintain a nice, big sound.

- *bend:* Sets the amount that the *pitch* output varies when MIDI pitch bend messages are received. It is calibrated in semitones from zero to 24. Yes, 24-count-em-twenty-four semitones, which really means a range of 48 (+- 24), or four octaves. Can you handle such power? No? OK, then set it to 7, or something.

- *glide:* Lets you bring a little or a lot of portamento (pitch glide between notes) to the party. Set it to the amount of time (in seconds) you wish Sumu to take when sliding between notes.

- *drift:* Applies your chosen amount of fluctuation to the pitch center of each voice. Keep things strait-laced and steady, or add some vintage smear to send a chill through your local ghosts.

- *mod #:* Lets you select which MIDI continuous controller signal to output through the Mod output. To use the mod wheel, set this to 1.

Outputs

- *pitch:* This output turns incoming MIDI notes into a pitch signal Sumu can use. When MIDI note A3 is played, the pitch signal output is 0. This has the same result on a patcher input as when nothing is connected. A4, an octave higher, outputs the value 1, and A2 outputs -1.

  Another 1 is added or subtracted for each octave up or down. This scaling was chosen so that keyboard input maps naturally to all the various control signals.

It's like the 1.0 volt per octave standard of some modular hardware, but there are no actual volts involved. So we can call this just 1.0 per octave.

In the patcher, input dials that can control pitch or frequency (such as *pitch* in *OSCILLATORS* and *level* in *GATES*) are calibrated so that when you connect a pitch input and set the default scaling (double-click), they will track the same frequencies or intervals according to the 1.0 per octave standard.

- *gate:* This output sends gate signals, with amplitude proportional to the velocity of incoming notes. The signals maintain their value after each key is released, allowing neat things like whacking on a drum pad at different levels of velocity to set filter cutoff over time, and such.

- *vox:* This output sends a signal proportional to the number of each active voice: 0.0 for voice 1, 1.0 for voice 2, and so on. This can be used to quickly make changes to the patch that are different for each voice.

- *after:* This output sends polyphonic aftertouch data for each key, added to the channel aftertouch value. There's nothing like routing aftertouch to a few parameters, and discovering a new dimension of control over notes you're already holding down! Really, do it. It's awesome.

- *dx* and *dy*: These outputs send continuously variable control signals, set by whatever MIDI CCs (continuous control) you selected in the hidden settings (available by clicking the three dots in the corner of the module).

- *mod:* This output sends a continuously variable control signal, set by whatever MIDI CC you've specified with the *mod #* dial above.

Channel aftertouch sends one value for the MIDI channel, and polyphonic (or poly key pressure) sends a different value for each key. Very few keyboards have true polyphonic aftertouch, so we decided these two kinds of aftertouch could share a signal output. If you don't have a keyboard controller with aftertouch, you can still use the output in Sumu by sending messages from a MIDI knob or fader controller.

- *time:* Whenever a voice is activated, this output sends out a signal that starts low, and climbs ever higher. This signal is perfect to patch into the *time* input on the *PARTIALS* module, to smoothly move the "playhead" for the current voice across the partials map.

PARTIALS

This module give you access to "partials maps" (please excuse the clumsy phrase—you'll be seeing it a lot). These are powerful scrolls of control data derived from audio analyzed by the included desktop app, Vutu. This is the key to all kinds of creative resynthesis fun when patched into the *OSCILLATORS* module, and a source of deep richness when modulating other parts of Sumu.

Audio is interpreted as 64 channels of pitch, amplitude, and noisiness values over time, one for each partial across the spectrum. This is displayed in a sonogram-like graph, with time moving left to right, and harmonic content visualized from top to bottom. Line weight implies amplitude.

Noise values, while also present and usable for each partial, are not shown on the display.

Each partial has a "playhead", representing its current place in time as a bright spot on the map. By default, all playheads are aligned in a vertical column, referencing the same moment in time for all partials. You'll see the column move back and forth when you vary the *time* dial, or when you play a note with the *time* output from *INPUT* patched into *PARTIALS*' *time* input. The playheads diverge from one another when you plug in more complex multichannel signals (as from *ENVELOPES*, *PULSES*, or from *PARTIALS* itself), each partial's place in time moving on its own.

You'll see multiple playheads per partial when more than one voice is active.

Choosing a partials map

A variety of useful maps are included. Open the partials menu under the map display to select from the available factory and user maps.

To import .utu map files you've created with Vutu, click the "..." button in the upper right of the module, and select "Import Folder." Navigate to your partials folder of choice, open it, and all maps inside will be available in the map selector.

Controls and related inputs

- *time:* Sets the starting point in time for the "playheads" in the partials map. When a one-channel-per-voice signal is patched in (such as *pitch* or *time* from the *INPUT* module), the playheads for all partials move across the map in a vertical column, one column per active voice. With 64-channel signals connected, the playheads can move independently. Signals entering the + input are added full-scale to the attenuverted signals entering the main input. Signals patched into the *x* input act as a multiplier for the signals entering the main input.

- *gate* input: If you want to inject a little sanity into things and bring this module into closer relations with incoming notes, patch the *gate* out from *INPUT* into this jack. With *loop* turned *on*, this lets loops play and repeat until you let go of the key, at which point playback resumes normally. Connecting a signal here also negates the effect of the *slew* control. All playheads warp back to their current starting point when a gate is received.

- *loop:* Switch on to make the playheads zip back to the other end of

the map when they hit the end. Turn it off to mercifully let them stop. Choosing ***gate*** mode turns on looping, only when you let go of a note, the playheads continue to the end of the partial map instead of turning off immediately. When the ***gate*** out from ***INPUT*** is patched into this module's ***gate*** input, looping only occurs while one or more notes are held.

To set the loop start and end points when a ***loop*** mode is active, click and drag in the partials map.

- ***slew:*** Lets you introduce some lag in the movement of the playheads. At zero, they update instantly when told to. This can come in handy for intriguing tape or vinyl-like time-scrubbing sounds.

- ***tilt:*** Sets the balance of noise amplitude across the spectrum. Turn left of center to emphasize noise signals associated with lower-frequency partials and dull those higher up. Turn right if it's Opposite Day where you are.

- ***blend:*** Boosts or cuts the intensity of noise that exits the ***noise*** output.

Outputs

As a starting point, try patching each of these outputs to the corresponding input on ***OSCILLATORS***. This will produce the baseline "additive resynthesis" behavior when a partial is loaded.

- ***amp:*** Outputs individual amplitude values for each partial at its current playhead position in the partials map.

- ***pitch:*** Outputs individual pitch values for each partial at its current playhead position in the partials map.

- ***noise:*** Outputs individual noisiness values for each partial at its current playhead position in the *you get the point, I'm sure*.

ENVELOPES

This module is (no points for guessing, Fred) an envelope generator. The fun part is, there are 64 envelopes inside. The output contains one signal for every partial, and they can differ (as we'll *get to if you'll kindly let me continue, Fred*). The display visualizes the relative shape and position of the stacked envelopes.

This module creates ADSR shapes, so your old pals Attack, Decay, Sustain and Release each have their own control.

To fire it off, send signals to its ***gate*** input, which pays attention to all 64 channels from the module you connect, each dealing with a separate envelope. It prefers sharp pings, such as ***gate*** signals from the ***INPUT*** module, or squared-off blips from ***PULSES***.

Other Controls

- ***scale:*** Choose ***vel*** to multiply all ADSR values by a value inverse to incoming note velocity. Louder notes make faster envelopes. Choose ***pitch*** to multiply all values by note pitch. Choose ***both*** to use both signals. Choose ***off*** if you want none of this.
- ***delay:*** Shifts the start time relationship between the envelopes, across the spectrum. The higher-frequency the partial affected, the more delay is added as you turn this up.
- ***t=0:*** Turn on when employing the ***delay*** dial, if you want to ensure zero delay is added to the lowest envelope. This makes the series of envelopes fire as soon as a note is struck.
- ***curve:*** Choose between exponential (read: snappier) and linear (read: un-snappier) envelope tendencies.

- *hi scale:* Scales the speed relationship between the envelopes, across the spectrum. The higher-frequency the partial affected, the more its envelope cycle speeds up as you turn this up. This can be handy for introducing a low-pass-style contour across the sound.

- *x vel:* Turn on to multiply all ADSR output levels by incoming note velocity. Louder notes make "louder" envelopes.

To attain "normal" velocity sensitive note volumes, patch this module into *GATES* and turn on this toggle.

- *level:* Multiplies all envelope output amplitudes by the value you choose.

Outputs

- *all:* The primary output, carrying all 64 channels of envelope signals.

- *lo:* A simpler signal, carrying only the envelope shape for the lowest partial, mirrored across all 64 partial outputs.

- *hi:* Another simple signal, carrying only the envelope shape for the highest partial, mirrored across all 64 partial outputs.

PULSES

On the right side, this module provides two synced pulse generators per partial, each with its own shape, rhythm, and dynamic options.

This wouldn't be *super* fun if you just hit them with a trigger and they went BANG every time, so on the left, you'll find a probability engine that lets you control when pulses should happen, for each partial.

It does this by applying shaped gradients of likelihood across the spectrum. This lets you make modulations that focus on a certain fre-

quency range. You could also, for example, use the pulses to trigger the envelopes all at different times.

At the top, you see a visualization of the current pulse probability shape for all partials, from lowest harmonics on the left to highest on the right. This will all make more sense as you stroll through the control descriptions.

Probability controls

- *hi scale:* Scales the variation in pulse rates for each partial across the spectrum. The higher-frequency the partial affected, the more its rate increases as you turn up this dial.
- *quant:* Controls how much the intervals between pulses are quantized. At zero, pulses fire right away when commanded to by the probability engine. At 1, pulses fire at 16th note intervals (four times the *bpm* setting, which can be interpreted as quarter-notes).
- *noise:* Adds noise to the pulse rate value, varying it randomly. This effect can then be reversed smoothly as you turn the control back down, returning rates to the default. The noise comes prior to quantization, so with *quant* full up, pulses still fire at 16th notes, but randomly shifted around.
- *shape:* Sets the basic form of the probability curve, with choices like "gauss curve" and "sawtooth."
- *free/sync:* Lets pulses run free, or rhythmically locked to the host clock.
- *bpm:* Adjusts the time interval between pulse cycles. By default, all pulse generators fire in sync (where probability allows) according to

this setting, but patcher input and/or ***hi scale*** goosing can differentiate pulse times per partial.

- ***center:*** Sets the center of the probability curve. Turn left to focus things on lower partials, and right for higher.

- ***width:*** Hear me out—this sets the *width* of the curve, letting you focus probabilities around the center point, or distribute them more liberally to either side.

- ***x prob:*** Lets you increase pulse probability, for all partials manually, or individually with patcher inputs. Heightened probabilities are still affected by the shape of the curve.

Pulse generator controls

- ***envs 1 and 2*** Sets pulse shape for each generator, with choices like ramp, rectangle, and exponential.

- ***rhythms 1 and 2:*** Controls both the number of output pulses per cyclke, and the pattern of each group of pulses. Setting these to different values for each generator gets you crazy polyrhythms.

For example: If you set this to "••-" then every cycle is divided into three output pulses, of which the first two play and the third is silent.

- ***amps 1 and 2:*** Sets the amplitude behavior for each generator. Set to ***full***, all pulses are the same level. Set to ***x prob***, the volume scale is multiplied by the probability curve. Set to ***x rand***, pulse levels vary per cycle, at random. Set to ***prob***, pulse levels are affected by the probability level for each partial, according to the curve.

- ***levels 1 and 2:*** Scales pulse level for each generator. Accepts multichannel control signals for differing settings for each partial.

Outputs

- *all 1 and 2:* The primary outputs from the shape generators, each carrying 64 channels of pulse signals.
- *lo:* A simpler signal from shape generator 1, carrying only the pulse signal for the lowest partial, copied across all 64 partials.
- *hi:* Another simple signal from shape generator 1, carrying only the pulse signal for the highest partial, copied across all 64 partials.

SCOPE

This module gives you a handy way to visualize all kinds of multi-channel signals. Patch something in from any module in the upper area to see what it's doing. 64 stripes show signal level per channel, over time.

To switch between the default scope view and an alternate "bar chart" view, hit the "..." button and change the mode.

OSCILLATORS

This module is the audible heart of Sumu—its powerful oscillator bank. All 64 partials get the building blocks of a basic FM synth voice. One "carrier" oscillator which creates the primary signal, plus a "modulator" osc that can shape and mangle the carrier oscillator's frequency curve (as well as acting as its own sound source, if you like). All this, and a noise generator, too.

All the juicy parameters here can be controlled individually per partial, under the influence of beefy 64-channel control signals from the modules above.

Controls

- *offset:* Sets a linear frequency offset between the carrier and modulator oscillators, distinct from the comparative offset introduced by the *ratio* control.
- *ratio:* Sets a comparative offset between carrier and modulator frequency. Handy for all manner of classic FM noises, in conjunction with raised *mod index* settings.
- *noise:* Adds noise to the mod oscillator's frequency.
- *mod index:* Sets the amount of influence the mod oscillator has on carrier oscillator frequency, scaled by *ratio*.
- *amp:* Scales the output level of the carrier oscillators.
- *pitch:* Sets the pitch of the carrier oscillators.
- *noise:* Sets the noisiness of the carrier oscillators
- *mod and carrier out level:* These two small dials on the border between this module and *GATES* set the output level of the mod and carrier oscillators. To hear "purer" FM effects, turn down mod level and let the carrier oscillators ring through alone.

The *amp pitch*, and *noise* inputs on the carrier side love to receive the same-named outputs from *PARTIALS*. This sets up the crucial pitch, level, and timbre relationship between the analyzed audio you choose and the resynthesized sound you hear.

GATES

The *GATES* module is a dynamic volume control, akin to the VCAs (Voltage Controlled Amplifiers) found in modular synths. Its inputs comes from the *OSCILLATORS* module, with the carrier and mod oscillator signals for each partial blended according to the small knobs between the

two modules. Its outputs are sent to the *SPACE* module. You send *GATES* control signals, typically envelopes, and it nicely increases and decreases the amount of input signals passed to its outputs. The control signals you send flow through a vactrol emulation, this time with a settable decay, which opens up a world of cool, percussive envelopes.

GATES is an important tool for sculpting the dynamic profile of Sumu's sound, and it's pretty magical beyond that. Magic comes into play when you flip it into LPG (Low Pass Gate) mode and synth-bongo all night long.

Controls

- *level:* This dial sets the static level of this module's level attenuator. Signals from the *level* input modulate the level, as well. For normal, keyboard-like playing, you'll probably keep this control at zero, so only incoming envelope signals triggered by key presses affect the sound's volume. For drones or reverse-enveloped sounds, try raising it higher.

- *lopass:* This toggle turns on low-pass gate (LPG) mode. In LPG mode, the gate's gain-changing cell is replaced by a low-pass filter, the frequency of which is modulated by the level signal. The modulation afforded by the vactrol emulation makes percussive envelopes with a very particular sonic signature.

- *decay:* Sets the decay constant of the vactrol algorithm. At low *decay* settings, the *GATES* will follow incoming mod signals very snappily. At higher settings, the decay of the vactrol rings out more and more.

SPACE

This module is, in the simplest terms, a panner and mixer for all of the partials. Each partial enters, and is placed in space according to rules you establish. Then, every single one of them can *move around* in whatever way pleases you most, and can be influenced by other modules, too.

To keep this from straying into "God's own giant Flying Faders console" levels of complexity, we provide two helpful tools—patterns of position and motion tendency we term ***home*** and ***field***.

Each ***home*** preset places each partial in different default positions within a theoretical 3D space, as visualized in the display.

In the end, each partial's position in 3D space is used to spatialize it across the stereo field.

Each ***field*** preset offers a distinct 3D vector field, providing a directional tendency for each point in that 3D space (x, y, z). This can lead to patterns of movement akin to tornadoes, waves, rivers, and so on.

Controls and inputs

- ***reset:*** Patch trigger-y signals into this input. Each blip resets the affected partial back to its default home position. Or hit the button to reset all partial positions.

- ***speed:*** Sets the rate at which each partial moves according to the directional tendency dictated by the current ***field*** preset.

- ***noise:*** Adds noise to the vector field, making partials move in jittery ways.

- ***theta:*** Swivels the 3d sound field around, taking all partials' positions with it.

- *x:* Offsets each partial's position on the X axis.
- *y:* Offsets each partial's position on the Y axis.
- *z:* Offsets each partial's position on the Z axis.
- *room selector:* Lets you choose between an *anechoic* space with no wall reflections, and a selection of rooms with natural reflections.
- *home:* Lets you select the home pattern of your choice; the starting place in space for each partial.
- *plane:* Rotates your chosen *home* pattern on its axis.
- *scale:* Changes the size of your chosen *home* pattern in space.
- *field:* Lets you select a vector field to affect the spatial position of each partial over time.
- *far:* Lets you choose what happens to partials when they go out of bounds. Choose *bye* to let them go very far away. Choose *Home* to make them return to their home position. Choose *flip* to make them re-enter the space from the opposite side. Choose *rand* to place wayward partials at a random position.
- *rotate:* Rotates your chosen *field* pattern within the space.

The output from this module is mixed down to stereo and handed off to *RES*. No more 64-channel madness from this point on.

RES

This module acts as an overall EQ and color-box for the big bad signals you create. It's an analog-modeled four-pole filter with serious character, especially when driven hard.

Controls

- ***freq:*** Sets the center frequency for the filter. The input on the left has a preset scale that corresponds to "1 volt per octave," allowing you to send ***pitch*** signals to the filter for tuned whistle-y fun. The input on the right has an attenuverter, opening up other uses.

- ***q:*** Adjusts the resonance for the filter, creating a peak in energy around the center frequency.

- ***mode:*** Selects the shape for the filter, with the options lowpass, highpass, bandpass, and thru (no filtering).

OUTPUT

This module lets you put the final polish on your sound before it's sent out into the cold, cruel world. Here, you'll find a master level control, a toggle-able soft clipping circuit, and a nifty little stereo oscilloscope that shows you the waveforms you're making.

Controls

- ***level:*** Sets the overall output level of Sumu. Note that this control also has an input, which lets it act like a VCA (voltage controlled ampli-

fier) when fed a control signal. This can come in handy for reining in sounds, such as those with a lot of filter resonance.

- *clip:* Turn this option on to soft-clip loud signals before they have a chance to clip the output in a way that sounds less nice.

  The soft clipping will only kick in audibly when the outputs from the *RES* module are quite loud. For example, most of the factory patches are set so that they sustain at approximately -12dB. It's not until playing four voices simultaneously that there will be enough signal to clip the output. With clipping off, this four-voice signal would be at 0dB. With clipping engaged, the sum will be smoothly turned down to around -3dB. Of course, if you have your favorite dynamics processor that's part of your trademark sound, by all means turn our clipper off and use your own. We won't be offended.

4 Vutu: Your Audio Cartographer

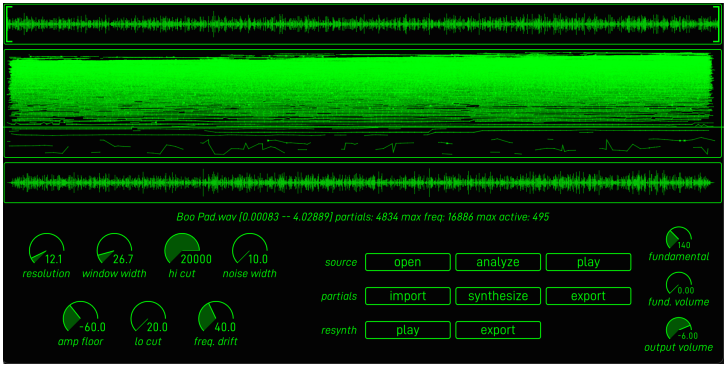
While Sumu's *PARTIALS* module is all about wringing the expressive synth-soul from your choice of recordings, you can't simply drop audio right into it like you might expect. Your files must first be analyzed. This happens *outside your host*, in a dedicated app provided with your copy of Sumu, called Vutu.

Finding Vutu

Vutu will be updated separately from Sumu, so it's a separate download. To get it, go to madronalabs.com, and make your way to the bottom of the Sumu page. There should be both Mac and Windows installers there. If you are technically able and enjoy pain and misery, you can also compile Vutu from its very source code. That code is available at github.com/madronalabs/vutu.

Charting the Course

All set? Now let's try out Vutu and see how it works.



For the most part, this is how you'll use Vutu:

1. ***open*** your source audio file
2. Temporarily turn up ***fund. volume*** to hear the pitch reference tone, ***play*** your audio file, and set the ***fundamental*** frequency to match your audio
3. Adjust the analysis settings to suit your signal, if needed
4. ***analyze*** the audio to create your partials map
5. ***synthesize*** your partials map back to audio
6. ***play*** the resynth'd audio, to see how it sounds
7. Rinse and repeat until you love what you hear
8. ***export*** the partials map and import into Sumu

The app contains a microcosm of Sumu's signal path. When you hit ***synthesize*** to prove out a new partials map, its pitch, amplitude, and

noisiness signals are used to drive a set of oscillator pairs just like Sumu's. The *resynth*-esized sound is captured so you can hear it, or even export it for use elsewhere, if you're so inclined.

Controls

- *resolution:* Sets the minimum frequency between analyzed partials, in Hertz. Thus, with a sound with a 100Hz fundamental, setting resolution to 100Hz or less would allow the harmonics at 100, 200, 300 and son on to be captured. Setting it higher would force the analysis to skip some partials, and make a weirdly hollow sound.

- *window width:* Sets the frequency in Hz of the analysis window. Higher frequencies tend to work better for rhythmic, transient sounds, while lower frequencies work better for sustained sounds.

- *hi and lo cut:* Excludes your choice of frequency range from the low and high end of the spectrum before analyzing. Can be used for cleaning up unwanted subs or grime from signals, or for focusing analysis on a certain frequency band.

- *noise width:*

- *amp floor:* Acts a bit like a high-pass filter, but for amplitude. The higher you turn it, the more quiet parts of the sound are stripped away.

- *freq drift:*

- *fundamental:* Sets the "home frequency" of the partials map you create, so its pitch info will play "in tune" with incoming note values

within Sumu. This control also sets the frequency of the built-in reference oscillator. Tune this while letting your source audio play, until the reference tone matches the fundamental pitch you hear in your audio.

- *fund. volume:* Sets the volume of the internal reference oscillator. Turn it up so you can hear it while setting fundamental pitch, then turn it down so you don't go crazy.
- *output volume:* Sets the output volume of Vutu, for your listening comfort.

IMPORTANT: Sumu can play back a maximum of 64 bandwidth-enhanced partials per voice. To reproduce your sound in Sumu, the maximum active partials at any one time in the exported analysis must be under 64. You can see this value in the status bar in Vutu, directly under the resynthesis display.

A Frequently asked questions

Why "Sumu?"

It's the Finnish word for "fog," which feels evocative of the sonic spaces it can create.

Where is it? I installed it but I can't find it in the Start Menu / Dock / Applications.

Sumu is a VST and Audio Units format plugin. To run it, you need a VST or AU host on your computer. Please see the Introduction to this manual for more info, and try asking on our web forums if you need advice finding a host to use.

Is Sumu supposed to sound like this?

Probably, unless you hear an abrasive series of glitches. Here's a good way to check that Sumu is functioning well: select the "default" patch from the factory patches section of the patch menu. If needed, turn up the ***voices*** control to get all of the voices running.

Now, turn the ***level*** dial in the ***GATE*** module up a little bit. You should

hear a mellow, slowly shifting drone. If there are any glitches in the audio, they will be readily apparent.

I hear glitches, how do I get rid of them?

The most common thing that needs adjustment is buffer size. Your host gives you a control somewhere over the size of the small buffers it fills up with calculations, over and over, to generate a steady stream of sound. If this buffer is too small, the calculation takes much longer, and even the fastest computer won't be able to keep up. Try turning the buffer size up to some number greater than 256. This should let Sumu run as fast as possible.

In Ableton Live, the buffer size control is under "Settings... / Audio / Buffer Size." For other hosts, it's probably something similar: please check your host's manual for details.

If the buffer size made no difference, it's possible that your computer is not fast enough to run all of Sumu's voices. You can try turning the ***voices*** control down to 1, and turning up the audio again on the default patch. If this helps, then it's almost certainly the case that CPU power is the issue. You can try adding voices one by one to hear where the problems come up.

If you are running the 64-bit version of Sumu in a 64-bit VST or AU host, you can expect to get around a 10% performance boost compared to the 32-bit version.

Finally, turning off animations with the ***Animate dials*** global setting or hiding the Sumu interface altogether will increase performance, for those times you are trying to squeeze out that last few percent and get your mixdown to happen.

Performance is affected by many, many variables including choice of

audio interface, drivers, host application and OS version. We can only give guidelines here. To tap into the collective wisdom of Sumu users on this topic, visit the ongoing discussions at madronalabs.com.

I bought one license for Sumu. Can I use it on my Mac and my PC too?

Yes. Sumu's license is very simple, but different from some you may have encountered. One purchase gives you a license for both Mac and Windows. You are restricted to running Sumu on one computer per license at any one time. If you want to run the software on more than one computer at a time, you must buy a licensed copy for each computer.

How does your copy protection work?

Sumu does not have copy protection. Copy protection always creates hassles for legitimate users. Our approach is different.

What we do is stamp each copy of Sumu securely with user data, consisting of your name and a unique ID. This is your own copy of Sumu, and you are free to make as many copies as you want. But do so carefully. When you run a copy, it may unobtrusively check to ensure that this data is intact and no other copies with the same user data are running anywhere. Since another copy running somewhere else could stop yours from running, we assume you will be careful about where your watermarked copies go.

We imagine, for example, that you might put a copy on your studio machine as well as your home machine, or on a USB stick to take to a mixdown session.

Can I load presets made in Sumu 1.0 in version 1.2 or 1.3?

Yes. Sumu presets will always be compatible with future versions, even

as we add controls and features.

On the other hand, if you try to load newer presets in an older version of Sumu, you will get errors.

I've got a Madrona Labs Soundplane controller. How do I set it up to work with Sumu?

Sumu detects your Soundplane's presence (provided you've got it plugged in and set up), and automatically switches its control behavior to make full use of the OSC/t3d control data Soundplane provides. Just plug in, and play.

I'm not playing any notes, so why is Sumu eating my CPU time?

Sumu has free-running oscillators and processors that are updated whenever your DAW is processing audio, so those eat up CPU at a rate that matches your chosen number of ***voices*** in the ***INPUT*** module. Just like on a modular synth, you can simply turn up the ***level*** dial on ***GATES*** to hear the oscillators, even if no notes are playing.

How do I make Sumu's dials change in response to MIDI data?

The ***INPUT*** module's ***mod*** output turns MIDI continuous controllers into continuous signals, which can then be sent to any destination in the patcher. This is a very flexible way of using MIDI controller data, because you can route and scale it quickly as a signal. The ***mod cc#*** dial sets the control number sent to the Mod output.

Sumu does not provide its own interface for changing a dial's position directly from a MIDI controller, often referred to as *MIDI learn*. Most plugin hosts, such as Live, Logic and Numerology, provide good interfaces for MIDI learn that work well with Sumu. Please consult the

manual for your host for details.