Welcome to Pure-Data...

Pure-Data is an open-source patching environment for multi-media (audio+image). Pure-Data is a programming language where you create relationships by connecting visual boxes (rather than typing complex commands). This lecture is separated into two sections, Concepts and Objects. We'll do concepts first followed by objects.

Click to enter each section:

pd Concepts	What is a patch/Object/Message?
pd Objects_1	message, print, Slider, Toggle, Bang, Number
pd Objects_2	metro, counter, openpanel
pd Objects_3	gemwin
pd Objects_4	gemchains, gemhead, colorRGB, alpha, square
pd Objects_5	rotateXYZ, translateXYZ, scaleXYZ, cube
pd Objects_6	pix objects (pix_texture, pix_image), sphere
pd Objects_7	putting it all together

When working with PD you are dealing primarily with objects, GUI (Graphical User Interface) objects, and messages. These are the building blocks of PD programming. When you connect objects, GUI objects, and messages you are creating a "patch". Patching is making something complex out of smaller building blocks.

You can think of a patch as plumbing. The way water flows through the plumbing of your house, messages flow through the connections in your patch. Objects change the flow of the messages to allow different things to happen. Messages always go into objects at the top, called the inlet, and always come out at the bottom, called the outlet. In PD messages flow from top to bottom.

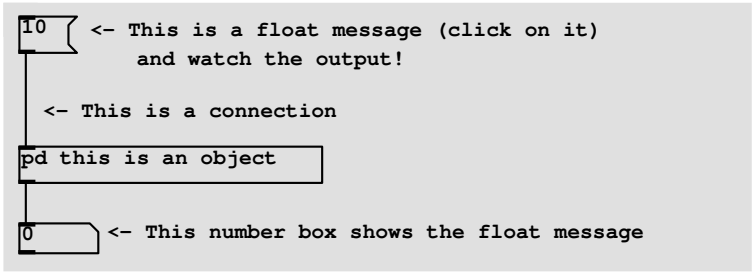
You can work with PD in two ways, the first is in "edit mode". Edit-mode is where you create your objects and the connections between them. "Run mode" is when you're done with the construction of your patch, and you wish to send messages through it. In run-mode your cursor is an arrow (as it is right now), in edit-mode your cursor is a pointing hand.

Objects are like filters, they change the way messages flow through them.

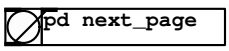
GUI objects allow you to interact with your PD patch as it is running. They allow you to change what your patch is doing without reconnecting the objects.

Messages are what allow objects to communicate with one another. Messages can change the way an object acts, and/or express the work the object is doing. Messages come in different types. They can contain words, numbers and groups of these. The main types of messages we will be dealing with are floats (numbers). You can click on a message, when in run-mode, to send it through your patch.

This is a very simple example of a patch, the message "10" can be sent through the "pd this is an object" and can be seen being passed out the outlet.



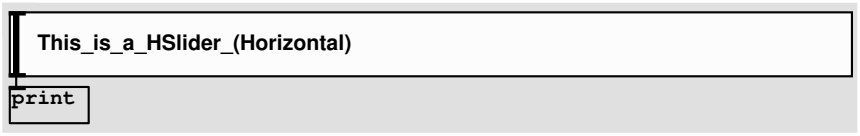
Click_on_Bang_For_Next_Page



All the commands you use in PD are objects. The behaviour of objects changes depending on what messages you send it. The first object we're going to learn is [print]. All print does is print out the messages you send it to the terminal:

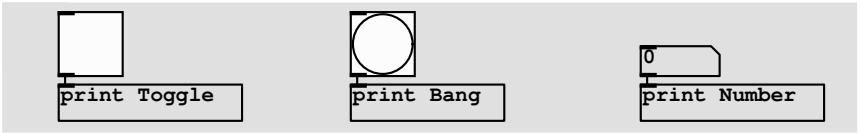
If you forget what an object does you can always double-click (on a mac) or right-click (on a PC) and then choose "help".

We will concentrate on two different types of objects: "objects" (of which print is an example) and GUI objects, (of which toggle is an example). GUI objects allow you to interact with PD, change parameters, etc. We are going to learn four types of GUI objects (but there are many more): Slider, Toggle, Bang, and Number.



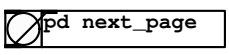
This HSlider is connected to the print object. This way we can see what messages the HSlider sends. Try clicking and dragging in the Slider. You can change the scale (and other properties) of GUI Objects by double-clicking (mac) or right-clicking (pc) and choosing "properties".

Slider, Toggle and Number all send messages made up of numbers. Bang is a special case and it only sends the message "bang". Below we're using an argument to the print object so that we can tell which GUI is sending which messages.

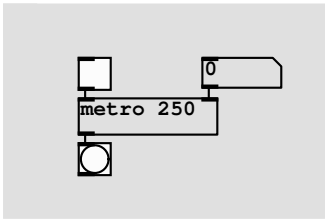


Toggle sends either 1 or 0, Bang always sends "bang" and if you click and drag on the Number you can see it acts a lot like a Slider. With Number you can also click once, and then type a number to send.

Click_on_Bang_For_Next_Page

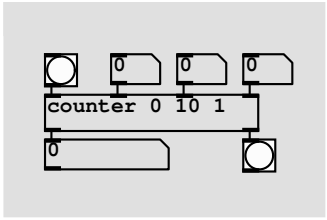


In this section we will learn three new objects, "metro", "counter", and "random". Metro sends a bang at regular intervals, just like a metronome.



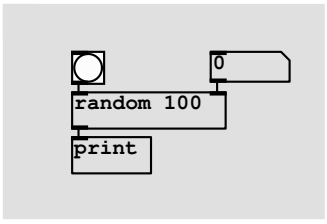
You can turn a Metro on and off by sending it a 1 or 0 message. Because a toggle sends 0/1 messages, we can simply connect it directly. Metro also accepts an argument (something after the object name). This argument is how fast the metro should send out bangs (in milliseconds). You can always change the speed of the metro by sending it number messages through the right inlet.

Counter is simply an object that counts. It can count up, count down and count up and then down. Where it starts and where it stops are all definable.

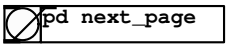


The first counter argument is the lower limit number (to start counting at). The second is the upper limit to count to. The third argument is the direction to count in. 1 means forward, 2 means backward and 3 means forward and then backward. You can also use the three right-most inlets to change the behaviour of counter. The right-most inlet is the upper limit, the second right-most the lower limit, and the third right-most as the direction. The rightmost outlet sends out a bang message when the counter loops.

The Random object returns a number between 0 and the argument when it receives a bang message in the left inlet. You can also change the upper limit by sending a message to the right inlet.

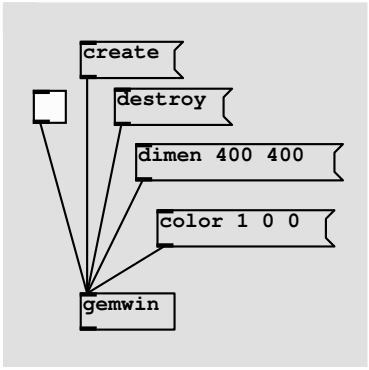


Click_on_Bang_For_Next_Page



Gem is the openGL (graphics) library for PD. Gem allows you to work with video/images and 3 dimensions all in real-time.

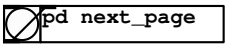
This section deals with the particular objects related to the Gem visual library. Gem uses PD Objects, as well as a set of Gem-specific objects. First I'm going to introduce the most important Gem object, the "gemwin".



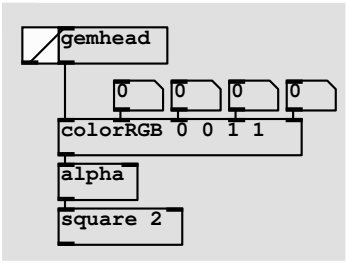
The gemwin object handles a lot of different messages. We're only going to discuss six of them for now. Gemwin is the object that deals with the Gem window. It handles messages that tell it to open the window (create), to close the window (destroy), to start rendering 1, to stop rendering 0, to set the size of the window (dimen), and to set the color of the window (color).

Just like objects have arguments some messages also have arguments. The "dimen" message expects two arguments, for the horizontal and vertical size of the window. Color expects three arguments - the amount of red, green and blue in the background color. Note that the RGB values accepted in Gem always go from 0-minimum to 1-maximum. You have to set the properties of the window before you open it with the exception of the color that can be changed after the window is created. The gemwin above creates a red window.

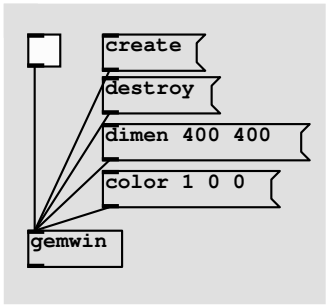
Click_on_Bang_For_Next_Page



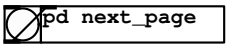
Gem Chains represent Gem visual entities. A single square in a gemwin is defined by a gemchain inside of PD. Gemchains always begin (at the top) with gemheads and always end (at the bottom) with geometries (circle, square, cube etc..). Between the gemhead and the geometry are the objects that change the entity's attributes. Here is a simple gem chain that defines a blue square. The Toggle connected to the gemhead tells PD whether it should render the square (1) or not (0). This way you can turn entities on and off in your scene.



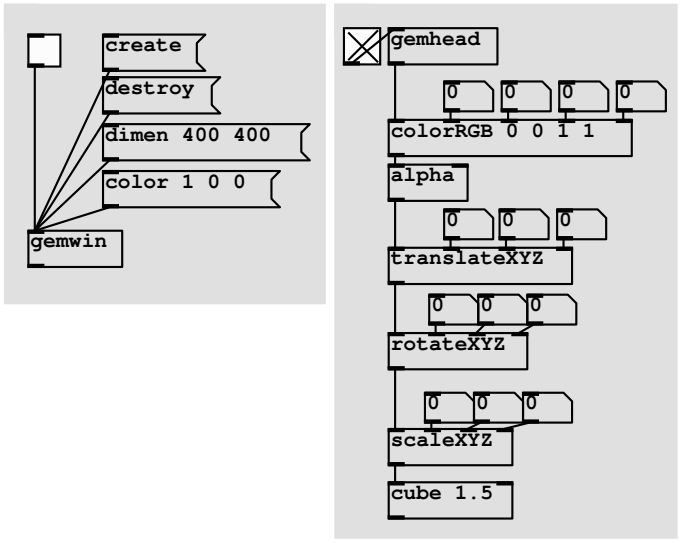
The "colorRGB" object changes the color of the entity defined by the gemchain. It takes four arguments, the colour (RGB) from 0-1 as usual and a fourth argument from 0-1 that defines the opacity. 1 is opaque and 0 is invisible. The opacity value of the entity is ignored unless the "alpha" object is present in the chain. Note that in order to use a numberbox with values from 0-1 you must hold down the SHIFT key as you click and drag. To play with this example create the window from the previous example: (Don't forget you have to turn on rendering to see the square!)



Click_on_Bang_For_Next_Page



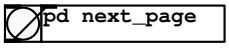
In this section we will learn three new gemchain manipulation objects. These are rotateXYZ, translateXYZ, and scaleXYZ, which rotate, translate and scale the object appropriately. Below is the previous example of a gem chain with rotateXYZ, translateXYZ and scaleXYZ added, and the "square" geometry changed to "cube":



All these objects have 4 inlets. The left inlet is to connect the gem chain and the three right inlets are for the X, Y and Z values for each manipulation.

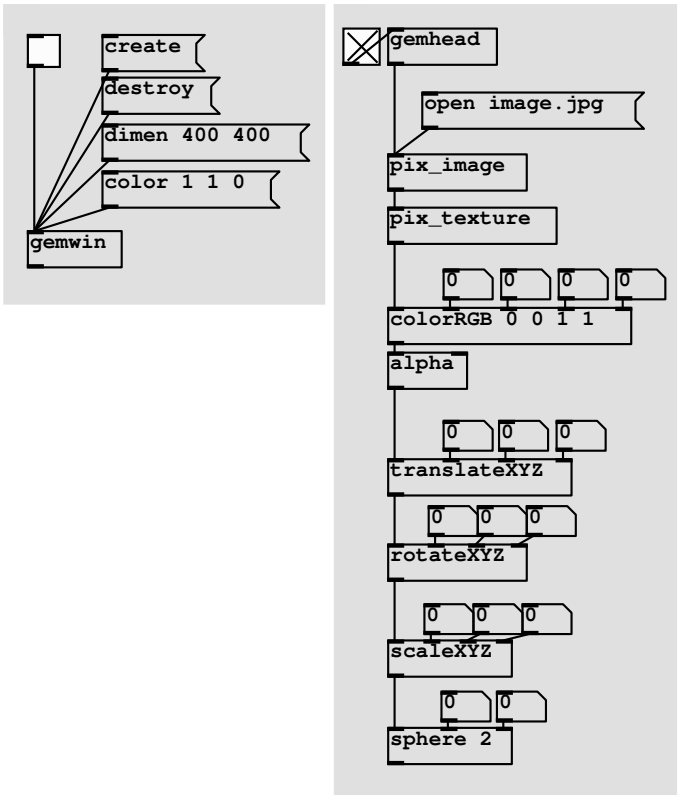
rotateXYZ accepts degrees of rotation. scaleXYZ and translateXYZ, on the other hand, use gem units. Gem units are arbitrary units that divide the gem space. The gem space is 8 units wide (x), 8 units tall (y) and 20 units deep (z). The center of the gem space is at (0, 0, 0). (This is the position in space on x, y and z.) This means that in order to put the entity in the upper left corner you need to send translateXYZ (-4, 4, 0) (-4 for X, 4 for Y, and 0 for Z).

Click_on_Bang_For_Next_Page



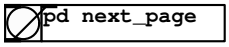
In this section we're learning a new set of objects that deal with images. The images that the pix_ objects deal with are applied as textures onto Gem entities. The pix_image object allows you to load an image-file. In order to tell it what image to load, it understands the "open" message. This message has a single argument - the filename of the image file. The image should be in the same folder as the PD patch. The pix_image object does not apply the image to the entity;
that is the job of pix_texture. pix_texture simply takes the pix data and applies it to the entity. To apply the image.jpg file to the sphere below click on the "open image.jpg" message.

The sphere object has one argument that is the initial size. Sphere has two additional inlets. The second from the right inlet also defines the size of the sphere. The right inlet defines how many sides the sphere has.

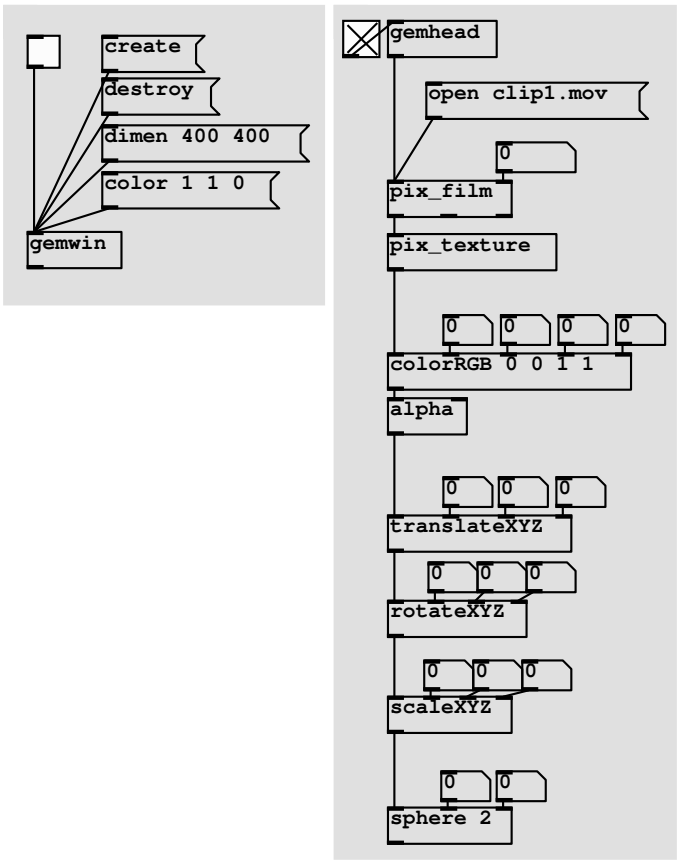


Tech note: You can load JPG and TIF files with pix_image. The dimentions of the image must be 2^x (for example 64, 128, 256, 1024)

Click_on_Bang_For_Next_Page



This is your opportunity to play around. I've modified the previous example so that rather than loading an image-file, you can actually load Quicktime files. pix_film accepts the same "open [filename]" message as pix_image. Click on the open message to open the Quicktime. The right inlet on the pix_film object defines what frame the Quicktime is on. You can use this to scrub the video. Try using counter and metro to control the color, translation, rotation or scale of the entity. You could replace any of the number boxes below with sliders (don't forget to change the scale!).



Go_Back_To_Index

pd index