This is a collection collection of threads during the FlyWithLua NG beta process to try to get all this information in a central location and hopefully in the end will be a manual.

Starting with version 2.7.0, FlyWithLua now supports creating native plugin windows. There are two ways to use them:

- graphics module: You can use the existing graphics module of FlyWithLua to draw into the windows, see e xamples included in the "Scripts (disabled)" folder that start with “floating_wnd”.

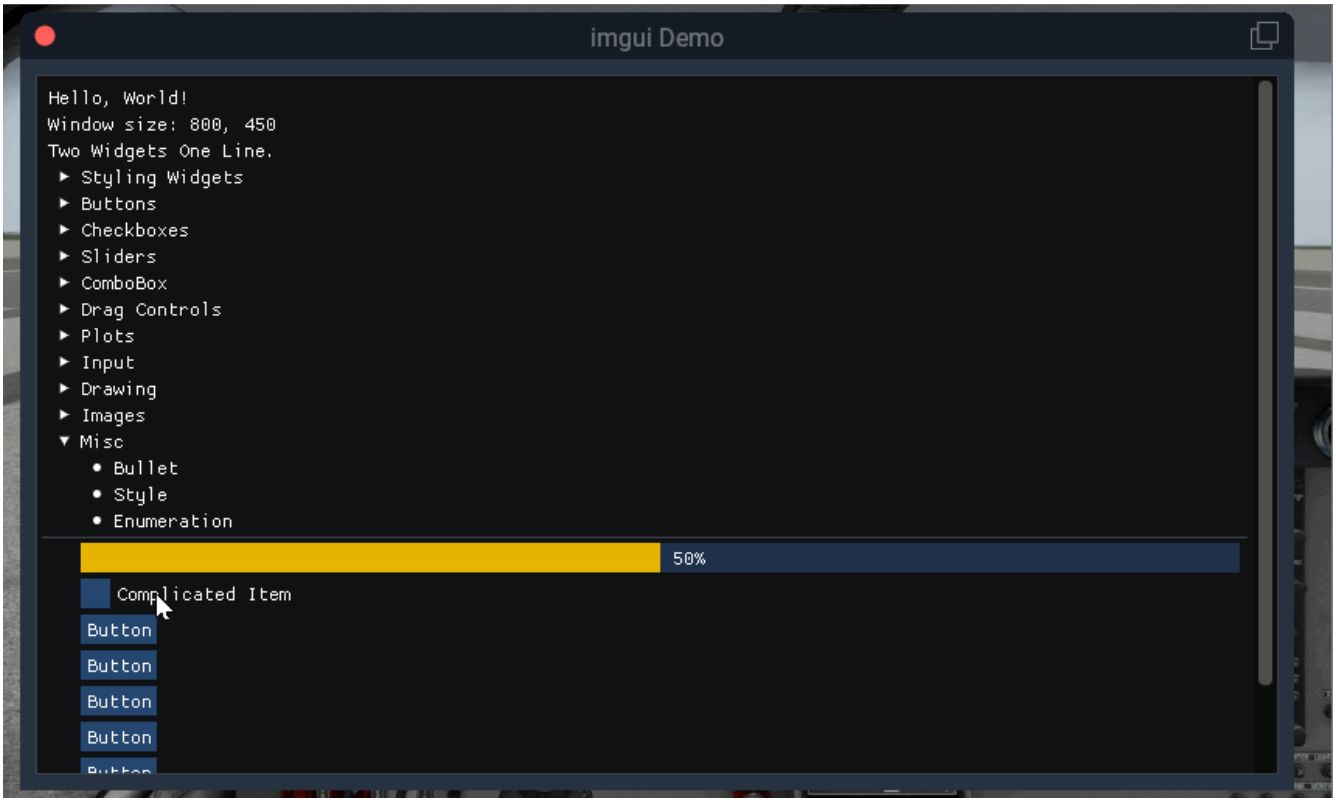
- imgui: FlyWithLua now supports creating imgui GUIs. imgui is an awesome 3rd party GUI library that makes it really easy to create stunning GUIs. See the examples included in the "Scripts (disabled)" folder that start with "imgui".

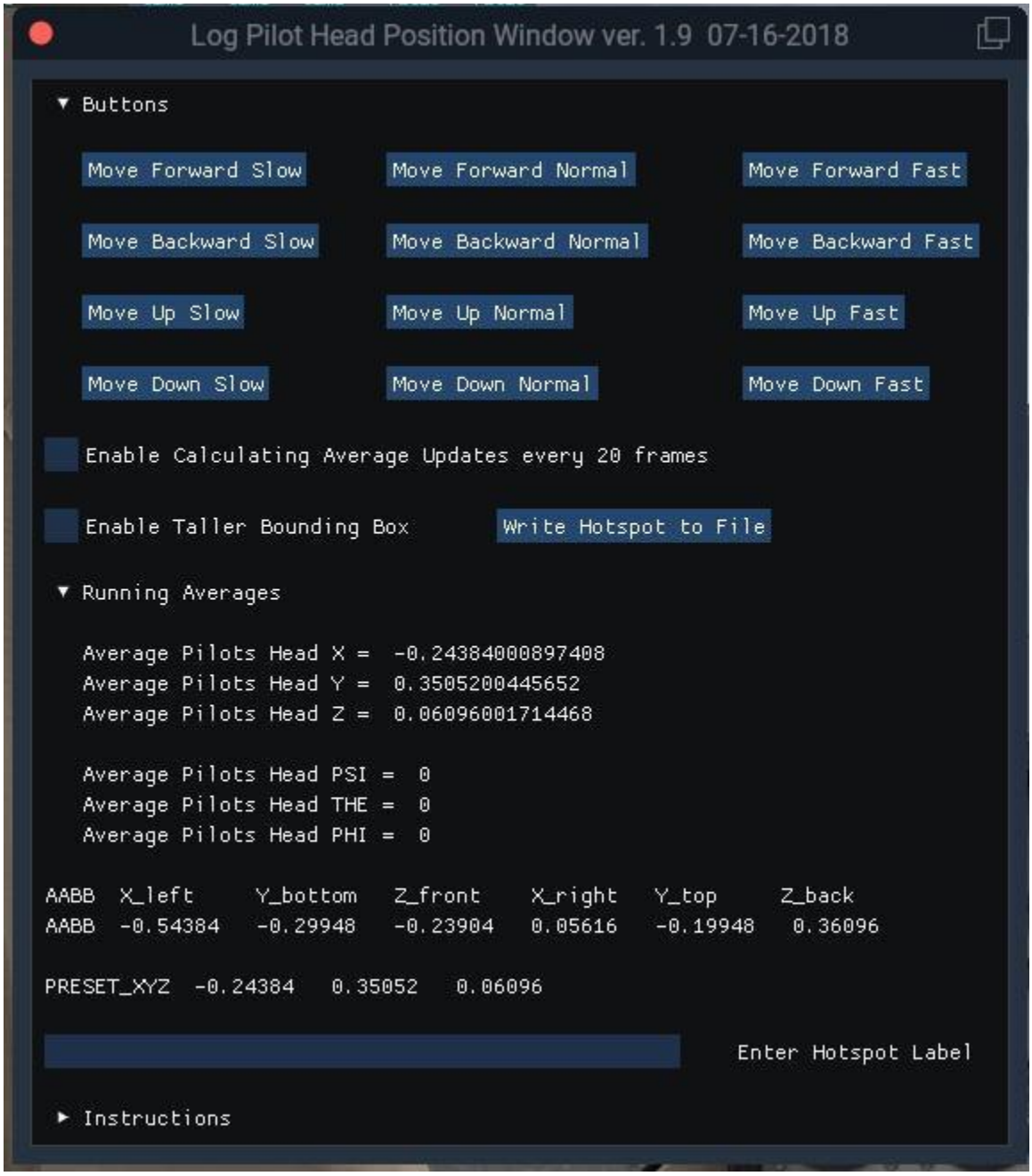
Why would you want to use these new capabilities instead of using existing GUI libraries written in lua?

- the windows can be moved and popped out to a native window
- the performance is better because the GUI isn't created by lua (for imgui windows)
- imgui is more powerful than the existing libraries - for example, you can load an image file (even png and jpg) and use it as a background image or button or whatever...
- both window types automatically (!) work in VR

Here are a few teasing images to see what is now possible.


FlyWithFPS
Clear Histogram
FPS
Histogram


imgui Demo
Hello, World!
Window size: 800, 450
Two Widgets One Line.
Styling Widgets
Buttons
Checkboxes
Sliders
ComboBox
Drag Controls
Plots
Input
Drawing
Images
Misc
Bullet
Style
Enumeration
50%
Complicated Item
Button
Button
Button
Button


Log Pilot Head Position Window ver. 1.9 07-16-2018
Buttons
Move Forward Slow
Move Forward Normal
Move Forward Fast
Move Backward Slow
Move Backward Normal
Move Backward Fast
Move Up Slow
Move Up Normal
Move Up Fast
Move Down Slow
Move Down Normal
Move Down Fast
Enable Calculating Average Updates every 20 frames
Enable Taller Bounding Box
Write Hotspot to File
Running Averages
Average Pilots Head X = -0.24384000897408
Average Pilots Head Y = 0.3505200445652
Average Pilots Head Z = 0.06096001714468
Average Pilots Head PSI = 0
Average Pilots Head THE = 0
Average Pilots Head PHI = 0
AABB X_left Y_bottom Z_front X_right Y_top Z_back
AABB -0.54384 -0.29948 -0.23904 0.05616 -0.19948 0.36096
PRESET_XYZ -0.24384 0.35052 0.06096
Enter Hotspot Label
Instructions

The following functions don't work with floating windows in VR anymore, no matter if using imgui or FWL: do_every_frame, do_on_mouse_click.

Instead of using do_every_frame, you can register a drawing function that is called everytime X-Plane draws the window. It gets passed the x and y coordinates of the window's lower left corner as *absolute coordinates*. This is required because FWL's graphics functions take absolute screen coordinates and you have to make sure that you actually draw into the window. For example, if you pass (0, 0) as coordinates to a lua function that draws a dot, the pixel would always be in the lower left of the *screen* because they are absolute coordinates. But if the window doesn't cover that part of the screen, the pixel wouldn't be visible in the window. So if you want to put the pixel in the lower left of the *window*, pass (x, y) as coordinates to the lua functions. If you want a button on top of the window with a padding of 10px from the borders, you would draw a rectangle with the first point at (x + 10, y + height - 10). I attached a sketch to visualize this. x and y are passed as paramters to your drawing function by FlyWithLua and you must use them to place your pixels inside the window.

And instead of do_on_mouse_click, you can register an onclick function with the floating window. That function is only called if the click actually happened inside the window, so it gets passed *relative coordinates* already. So if the user clicks on the upper left of your button's rectangle, it would get passed (10, height - 10). You would then have to set a flag to remember that the user clicked somewhere so that you can then check this variable the next time the window is drawn.

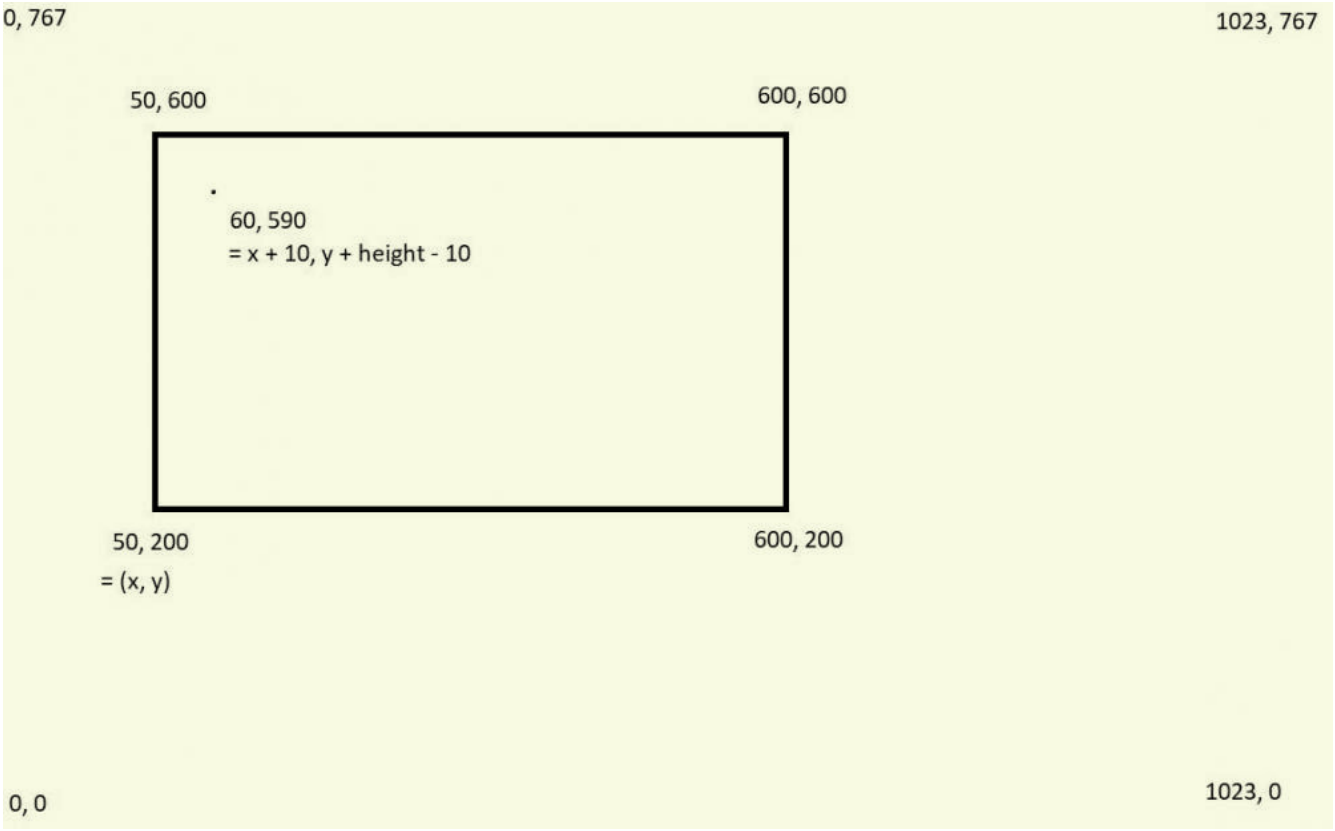
Using imgui, you don't need to care about drawing the controls yourself or mapping the clicks back to the controls. In fact, you don't need to register an onclick function at all. imgui is an immediate GUI, that means the controls don't have any internal state. You can't ask a checkbox if it is currently checked and you can't set a callback function to a button. They just have no state at all. How does it work then? For imgui windows, you register a builder function. That function is also called everytime the window is drawn. If you want to place a button, you just call

```
imgui.Button("My Button")
```

When you do that, imgui does not only draw the button, it also checks if the mouse button was released over this area in the last frame. And if that's the case, the Button function returns true! It's a little weird to get used to in the beginning, but all imgui functions actually return boolean values to indicate whether they were just being used. So if you want to react on the button, you just do something like this:

```
if imgui.Button("My Button") then
  command_once("some x-plane command")
end
```

In the next frame, the mouse will not have been released over this button again, so the button doesn't return true in the next frame, resulting in your action being only called once as desired. The same goes for other controls like sliders and checkboxes - they return a boolean to indicate if you need to react on a change.


0, 767
1023, 767
50, 600
600, 600
60, 590
= x + 10, y + height - 10
50, 200
600, 200
= (x, y)
0, 0
1023, 0