From: **seanharre** seanharre@gmail.com
Subject:
Date: February 25, 2015 at 11:24 PM
To:
Bcc: seanharre seanharre@gmail.com

S

First, requirements and I think this is best where I help get you going,

- download Leap SDK (you will need to subscribe) and install .dmg
- download OpenFrameworks v0.8.4, put somewhere (my source code gets placed inside at apps/myApps/3dglove)
- read `3dglove_INSTALL_HOWTO.txt`

This code uses OpenFrameworks for graphics. It's very similar to Processing (they used as basis) but has more features and is written in c++.

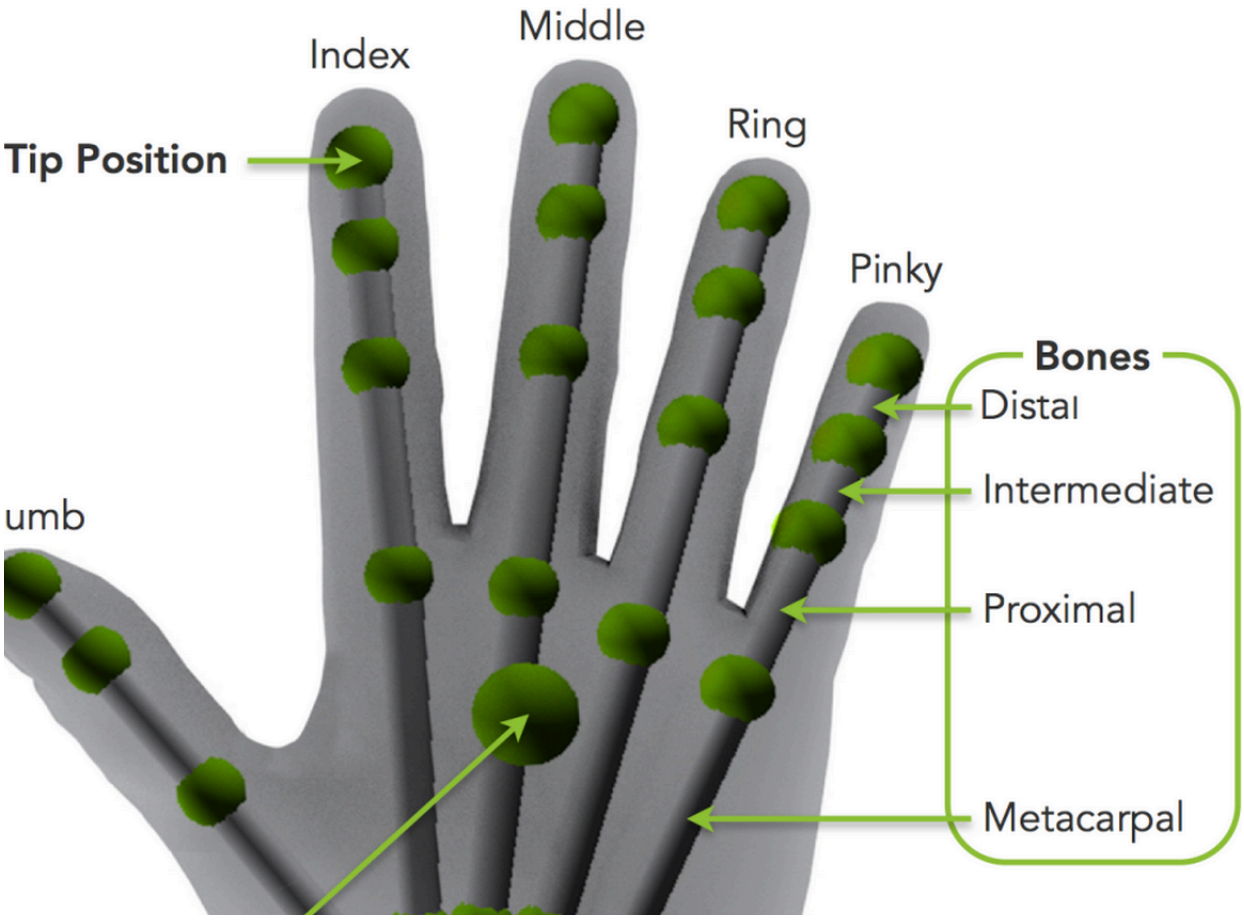
OpenFrameworks (ofApp.cpp) main flow is to call ::setup() at boot, then it just loops calling ::update() then ::draw(). In my code, it is just handling getting graphics data from the thread and displaying. That's it - it's just displaying what's going on under the hood.

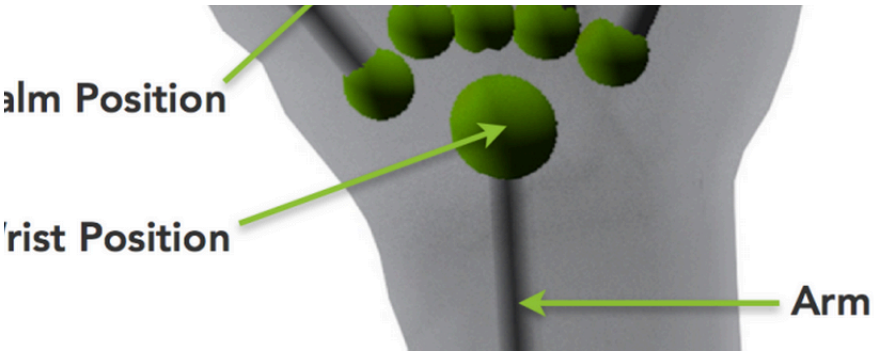
In that ::setup() I call TW.start() to start the thread worker who does everything else. It starts the Bullet physics engine, Leap motion controller, sound ctrl, and FTDI USB cable ctrl if you have them. The thread sets up a listener to Leap (for when it has new data) and gives callback to physics (called as it simulates/steps the world forward).

All build config is stored in 3dtarget.h. I will give you a Leap controller but since you won't have FTDI cables, set yours this way,

```
// compile cfg
//----------------------
#define CFG_HAS_RIGHT_HAND      0   // has FTDI USB plug for right hand
#define CFG_HAS_LEFT_HAND       0   // has FTDI USB plug for left hand
#define CFG_HAS_LEAP_CTRL       1   // has Leap Motion controller
```

So the big picture is the Leap controller listener calls ::onFrame() at about 100hz to give new hands data. Both left and right are included if they are in the field of view of the Leap, and each hand has 24 data points at joints.





For every new hand dataset, I push those new points into the physics engine and step it forward at a 200Hz step interval. (The bullet engine is keeping track internally of how long ago it was last stepped forward so it knows how long it needs to simulate at each invocation and uses the passed step interval to tell it max step size it can use).

After bullet has simulated the world forward one step, it calls ::bulletTickCallback(). This callback sets a global so the thread knows it has something to do.

When the thread has something to do, it updates any impulse/collision information for each joint from bullet, makes impulse sounds, and writes the new data out the FTDI cables to glove hardware if available.

As I mentioned, some places I would like help would be in the physics engine. Two immediate issues I see,

- currently grabbing a sphere is difficult as it wants to shoot out of your hand
- soft body impulses are not being realized (i.e. no sounds are generated when you grab the soft blob)

I'm thinking adding a soft-body around the rigid-body sphere and see the response. Is it easier to grab?

But then we aren't yet getting impulse data from the soft body, so the user will feel no interaction when grabbing an object enveloped with a soft body.

So I think that's really the first item to be fixed is to dig into the physics engine and get the impulse data when soft bodies are grabbed.

One demo I'm thinking about is playing tetris when you can grab the blocks, twist them around, and throw them down. For that kind of interaction, having a soft body around a rigid body would allow more realistic interaction.