MPX TUIO Driver

Google Summer of Code 2009 Proposal
Mentorship: Natural User Interface Group
Student: Ryan Huffman

Abstract

The current state of TUIO implementations require the use of explicit reception of TUIO messages, rather than adhering to a generic multi-touch interface/API. Unfortunately, given the general adolescence of the technology in the consumer world, there hasn't been a generic way to do this using a respective window manager's API. If a generic interface were in existence, applications could be written to be MT compatible, instead of exclusively TUIO-compatible, or [other-MT-device]-compatible.

With the creation of Multi-Pointer X (MPX), the possibility of creating Multi-Pointer native applications is possible. To utilize these features with the currently existent TUIO trackers, though, we need a way for MPX to interface with these devices. To do this, an xorg-input-tuio driver will be written to receive TUIO events, analyze them, and pass them along as MPX blob events.

The integration of TUIO based hardware into MPX means that not only will native applications be able to be built with MT capabilities, but an MPX-enabled X-server coupled with the MT capabilities of a TUIO device will be able to interact with legacy applications. Although new MT capabilities can't necessarily be added to an application that doesn't support it (gestures such as two-finger swipe scrolling could be added, though), it's single mouse/keyboard interface will be accessible, and there will be the ability to use multiple legacy applications in a multi-user environment - e.g. two users simultaneously browsing the Internet in separate windows.

In addition to the basic driver itself (xorg-input-tuio), several other features will be considered for implementation:

- Cropping of TUIO input so that input outside of a certain area is ignored.
- Joining of TUIO trackers. This would allow two trackers to appear as one. One example might be two trackers, representing adjacent surface areas. These areas could be joined as one.
- Setting the effective area of blobs. If, for instance, you had 4 trackers, you could set them to effectively use 1 quarter of screen space each. In use with joining TUIO trackers, this could lead to an ad-hoc way to do camera stitching (Having finger drags that go across two trackers may be tricky though, and should probably be left to the tracker). This is essentially transformation of the blob positions to screen space.

The set of actual features can be extended, and a final list would have to be made once development has begun and limitations are known. Once the driver starts to come together, it would be great to get community input on what they would like to see as well.

In order for this project to be ultimately successful, I think it is is extremely important that I stay involved with the community through each phase of development. Although writing the software and creating the "product" is what the project describes and intends to accomplish, it is useless without the eventual acceptance and use within the community. To support this, I will be be involved in the community by providing help and assistance through communication on the forums and on irc (in #nuigroup on Freenode), and by writing tutorials and/or examples on my blog.

Timeline

This is a preliminary timeline. Details would need to be discussed with my mentor once being chosen.

Phase 0 (April 20th – May 22nd) - Get to know the NUI community better. Get better acquainted with X.org and their community. Set up a home computer with an MPX enabled X server that will be dedicated to testing. Some time will also be spent researching the project.

Phase 1 (May 23rd – May 29th) – Outline the necessary components of the driver. Discuss the implementation of the driver and how some of the advanced features will be implemented with my mentor. This will, for the most part, be a design phase.

Phase 2 (May 30th – June 5th) – Begin writing the driver. The first priority will be receiving TUIO data from one tracker. Once this works, the data will be packaged and sent off as X blob input events. A test program will be written to display the X blob events. This should take about 1 week total.

Phase 3 (June 13th – June 26th) – Next, having multiple trackers at one time will need to be sorted out. Depending on the approach used, I am expecting this to take around 1 to 2 weeks, including testing.

Phase 4 (June 27th – July 5th) – Implement extra features of the driver. This will take around 1 to 2 weeks, and will likely use a portion of the mid-term evaluation period.

Phase 5 (July 6th – July 13th) – Mid-term evaluations. Continue working on project.

Phase 6 (July 14th - July 20th) – Test extra features.

Phase 7 (July 21st – July 31st) – Write CLI application to manage and configure the driver. This will allow a user to look at a list of the current trackers and other miscellaneous data. In addition to this, the application will be able to be used to modify driver features and to manage the trackers.

Phase 8 (August 1st – August 7th) - Adapt CLI application to a GUI interface.

Phase 9 (August 8th – August 17th) – Test CLI and GUI applications. Do a complete run through of all of the code written and clean it up if needed. Write documentation and how-to's for both users and developers.

Phase 10+ (August 18th – Later) – Rest. Then participate in more MT related festivities.

Details

There hasn't been much work to create an MPX TUIO driver to this date, probably due to the fact that MPX isn't in a release version of X.org yet; MPX will be in the next release

version of xorg, though.[1] There was one mention of TUIO support in MPX on Peter Hutterer's (creator of MPX) MPX blog that mentioned some work on it, which essentially received forwarded TUIO blob data to the xf86-input-blob driver.[2] Although this is a good start, it doesn't fully implement a TUIO driver.

What I am proposing is a TUIO xorg driver (xorg-input-tuio) written in C, that will receive TUIO events from a tracker and send them off as XBlobEvents. It would then only be a matter of writing an application that can receive these special events to enable MT functionality. One possible solution for receiving the TUIO events is to simply open a socket on port 3333 and listen for TUIO UDP packets, all within the driver. Unique devices could be distinguished by their IP/Port, and would be stored and managed by the driver. There may need to be some sort of timeout for deciding whether a TUIO tracker is no longer in existence, so that the list of trackers does not continually grow without ever shrinking. Merging trackers, as described above, would simply be a matter of modifying the blob id's of incoming TUIO messages from the associated trackers so that they do not clash. Cropping TUIO input would essentially require specifying rectangular boundaries for input, and having the driver ignore any input outside of that area. Handling the effective areas of a tracker would happen when coordinates are translated to screen coordinates, and would just be a matter of adjusting this scaling factor on the X and Y axes.

The CLI application would be written in C and allow a user to list the current trackers associated with the driver. Other options for the CLI application would be to set the features of the trackers and the driver itself. The GUI interface to this application would probably be written in C++ using GTK, but I am fairly flexible on this and could do it using Java or another language + GUI library.

Personal Information

Name: Ryan Huffman
Email: ryanhuffman@gmail.com
Location/Timezone: Sacramento, California / Pacific Standard Time (GMT – 7:00)
Website: huffmanr.wordpress.com
Age: 22
Education: 4th year Undergraduate Computer Engineering student at California State University Sacramento

I am very interested in getting involved with research in HCI and developing new methods for creating a more intuitive and immersive experience for the end-user. I love working with both hardware and software, and I am looking forward to building my own FTIR table in the near future.

I have been programming in C for around 2.5 years now, for both school projects and personal projects. The most notable of which being a simple OS for an advanced operating systems class. Although I don't have any open source development experience, I have been wanting to get involved with an open source organization for quite awhile. I use a lot of open source software myself, and use the Arch Linux distribution as my main operating system.

One of the goals of GSoC is to bring new developers into the open source community, not just a couple new projects and several good-bye's. At the end of the day, you want

1 http://www.x.org/wiki/Releases/7.5
2 http://wearables.unisa.edu.au/mpx/?q=node/104

developers and contributors that are going to become a part of the community, and be around long after the summer is over. I am extremely excited about participating in GSoC with the NUI Group, and I can tell you now that I will be a present member of the community whether I am accepted or not. With this in mind, I would love to dedicate my summer to furthering the integration of TUIO based MT devices into MPX.

If there are any questions, please don't hesitate to contact me through email. Thank you for considering my proposal!