Kivy – A Framework for Rapid Creation of Innovative User Interfaces

Mathieu Virbel[1], Thomas Hansen[2], Oleksandr Lobunets[3]

mat@kivy.org[1]
thomas.hansen@gmail.com[2]
alexander.lobunets@gmail.com[3]

1 Introduction

The software toolkits and programming paradigms used by application developers have a large impact on the design of architecture and user interface of the applications they develop. Especially now, that technological advances are presenting us with affordable hardware that is capable of both sensing and displaying information with unprecedented accuracy, vividness, and bandwidth, there is an increased need to explore and develop tools that enable creative designers and developers to turn their ideas into reality. These tools must both allow for rapid prototyping to evaluate new ideas quickly, and be able to take advantage of the full computing power offered by cutting edge technology. In this paper, we present Kivy (http://kivy.org), a software toolkit that is the result of our continuing efforts to explore this domain and create a powerful tool for developing the next generation of user interfaces. We give an overview of Kivy's architecture and design choices, and briefly showcase some of the first applications being developed in an effort to highlight the issues and opportunities involved in writing software Natural User Interfaces (NUI).

2 Motivation

The primary motivation behind this work was the lack of a cross-platform toolkit for programming Natural User Interfaces. We believe there is a possibility to merge all the approaches into one a single cross-platform powerful and developer-friendly framework. There are two key aspects, which define capabilities of a NUI framework: input handling and output rendering. We have been exploring the new event models, which are not implemented in other toolkits, paying attention to such features as an ability to store data inside the touch event, or to extend the touch event from the low-level hardware capabilities. In any classical

toolkit you will always have a position, maybe a blob size, a pressure value and an angle. Other data, which includes acceleration, blob image, relation between touches, hand/palm information, or other contextual information, can also be part of the event if it is computable from the data the hardware is capable of sensing. Another aspect of a powerful NUI toolkit is how it takes an advantage of modern graphics hardware, such as OpenGL ES 2 (http://www.khronos.org/opengles/2_X/). Using modern OpenGL programming allows us to have high-end performance, while it can be run on virtually any device, including mobile phones. We could find in any available frameworks both advanced input handling and efficient output rendering, not mentioning other developer-friendly features. Therefore we have created Kivy, a Python (http://python.org) / OpenGL ES 2 framework, adapted to create NUI interfaces, and run on Windows, Mac OSX, Linux and Android. Kivy has been released under the LGPL: you can freely use it for open-source or commercial product.

3 Related works

Kivy is an evolution of the PyMT (Hansen et al., 2009) project: the graphics engine and widgets library have been completely rewritten in order to provide much better rendering performance and to introduce property bindings that connect object properties to graphics primitives. The API has been normalized; inputs, core providers and events model have not changed, with the exception of adding new ones for Android support. There are several cross-platform toolkits similar to Kivy: MT4j (http://mt4j.org/) based on Java technology, TISCH (http://tisch.sourceforge.net/) written in C++, GestureWorks (http://gestureworks.com/) and Open Exhibits both based on Flash/ActionScript, Libavg (http://libavg.de) same as Kivy written in Python. Each of these toolkits is based on different technologies and has its advantages and disadvantages.

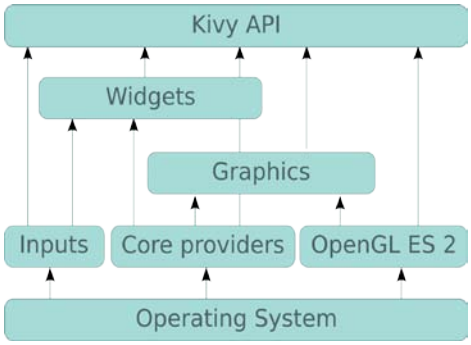

Figure 1: A general architecture of Kivy.

There are also a number of platform-dependent frameworks in the area of NUI. They include but not limited: Microsoft Surface SDK, WPF, Snowflake for Windows, and DiamondSpin for the DiamondTouch. Kivy's main difference is that we have focused on having a fast 2D

graphics engine based on OpenGL, implemented the abstraction layer for different devices and OS support, and delivered multi-touch ready events and widgets. We are still working on implementing good gesture engine (mono and multi-touch) and a native 3D support that does not require from application developers going deep into the OpenGL programming specifics.

4 System architecture

The Kivy framework is written in Python and Cython (http://cython.org/). Kivy uses Cython to accelerate all the components related to graphics rendering, matrix calculation, and event dispatching and properties operation. A general architecture of Kivy is depicted on the Figure 1. As you can see, by using Kivy API, your application will be neither OS nor device dependent, and will be able to run on any platform or devices. You can use the high level widgets, or do your own graphics using low-level abstraction.

Core Providers is the layer between the operating system and the library. Instead of using one specific library for one or multiple task, we made an abstraction for every core features like windowing, image, video, camera, text, spelling etc. If you use an Image class from the core providers, it will use the best one on the current operating system. All the widgets are using this abstraction, which makes them OS independent.

Every operating system and SDK has its own approach in terms of how to handle multi-touch, and lot of them is focusing on multi-touch events only. We made a global MotionEvent that can handle almost any type of input devices and OS event. Kivy has native support for Windows touch and pen event (WM_Touch and WM_Pen), Mac OSX, Android (through SDL/Joystick API), Linux (LinuxWacom, MTDev, and MT Protocol A/B from Kernel), and TUIO (Kaltenbrunner et al.). All of them will generate an instance of MotionEvent. But the instance can differ between implementation. MotionEvent is extensible through profiles: a profile is a string ID to indicate what kind of features actually supports. Each profile extends the touch instance with new arguments that you can use it after-wise.

The graphics engine is written completely in Cython. Python is great for rapid development, but we wanted to have the maximum performance with the same flexibility. To be able to push Kivy on mobile device, we have restricted ourselves of using non-OpenGL ES 2 instructions inside our graphics part. One of the features of the rendering engine is its capability to detect when an instruction have changed.

The 2D Widget class supports collision detection, touch event dispatching, custom widget events, event observers, and canvas. The widget interaction is build on top of events that occur. If a property changes, the widget can execute assigned logic. Unlike the other frameworks, we have separated the Widget and its graphical representation. This is done on purpose: the idea is allowing developers to create own graphical representation outside the Widget class. This separation allows Kivy to run an application in a very efficient manner.

5 Use cases

Kivy is actually used in an educational experimentation at Erasme (http://erasme.org) in Lyon, France. Erasme is experimentation center that focuses on the new usage of technology in the future, for college, museum, city and senior peoples. Kaleidoscope (http://github.com/tito/Kaleidoscope) is one experiment to allow the creation of learning sequences scenario that involves the individual and collective dimension, and the division of powers. The project involves a server that has all the scenarios, and displays the common part on a table. Children have a client tablet to interact with the table. Server is on Linux and client is on Android, using Motorola Xoom, Samsung Galaxy Tab and more.

One of our intentions was to demonstrate how a Kivy application could run across various devices. A PreseMT (http://github.com/tito/presemt/), a presentation application that uses multi-touch and pen-based input, has been created as a result. The editing mode allow you to drag text, image or video on an infinite plane, and scale / translate / rotate them. Then, you can adjust your view, and save it as a slide. On this mode, we are using one and two fingers interaction, double tap. In the view mode, you have button to go to the previous or next slide, but you can also scale / translate / rotate the infinite plane. You can use the pen for doing annotation, and double tap for removing all the previous annotation. The project is Open-source, and source code is available on Github. A beta release is actually available on Android market too.

6 Conclusions and future work

Kivy have been accepted inside the Google Summer of Code (GSoC) under the Nuigroup (http://nuigroup.com) umbrella. The current GSoC focus is providing a better Mac OSX support by using native library on OSX inside Kivy's Core Providers. iOS port is also under development. We are also looking into different implementations of the graphics compiler (Tissoires & Conversy, 2008; Tissoires & Conversy). We have also some concerns about porting Kivy to the Web platform. The idea would not to rewrite Kivy library in JavaScript, but have a way to compile a Kivy application to HTML5/WebGL/JavaScript code, similar to what Google Swiffy (http://swiffy.googlecode.com/) project does but for Adobe Flash .swf files. After learning and deeper analysis of PyPy (Rigo & Pedroni, 2006), it could be possible to achieve that point by converting our Cython part to pure Python code, and then translate all our python and application code to JavaScript. Then, we need to implement one of each core providers into JavaScript, and implement inputs providers for web-browser multi-touch, or supporting external NPAPI for OSC/TUIO (Lobunets & Prinz, 2011).

Acknowledgments

We would like to thank all the PyMT team for their experiments, and all the Kivy contributors especially: Gabriel Pettier, Rosemary Sebastien, George Sebastien, Joakim Gebart. This work have been supported by the following companies and organizations: Erasme, Nuigroup, Rentouch, Tabtou; and by the Google's Summer of Code Program.

References

Hansen, T., Hourcade, JP., Virbel, M., Patali, S., Serra, Tiago., *PyMT: a post-WIMP multi-touch user interface toolkit.* ACM International Conference on Interactive Tabletops and Surfaces, Banf, Canada, 2009.

Kaltenbrunner, M., Bovermann, T., Bencina, R., Constanza, E. *TUIO – A Protocol for Table Based Tangible User Interfaces*. Proc. 6th International Workshop on Gesture in Human-Computer Interaction and Simulation, Vannes, France.

Lobunets, O., Prinz, W. *Exploring cross-surface applications development using up-to-date Web-technologies*. Submitted to Interactive Tabletops and Surfaces 2011, Kobe, Japan, 2011.

Rigo, A., Pedroni, S. *PyPy's approach to virtual machine construction.* Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Portland, Oregon, USA, 2006.

Tissoires, B., Conversy, S., *Graphic Rendering Considered as a Compilation Chain*, ENAC, Toulouse, France, 2008.

Tissoires, B., Conversy, S., *Hayaku: Desigining and Optimizing Finely Tuned and Portable Interactive Graphics with a Graphical Compiler.*