

Easy Mobile Augmented Reality using Python

Cristiano Lemos Soares

HIT Lab NZ,

University of Canterbury

cristiano.soares@canterbury.ac.nz

Abstract

Advances in mobile technology, including the constant increase in the processing power of mobile devices and in connectivity have turned mobile devices into a powerful platform for Augmented Reality (AR) applications. Thanks to this evolution, different technologies, such as AR, Global Positioning Systems (GPS), 3G connectivity and Bluetooth can now be used on mobile devices to capture, present and share AR data. Despite the infinite advantages of using these features, the development of AR applications has become an increasingly complicated task. The use of scripting and interpreted languages, such as Python, simplifies this task, making it easier to prototype advanced AR applications. This paper presents how developing a Python wrapper for common performance-critical software modules in Symbian C++ can increase the speed of development of such applications. Augmented Reality programs then become much shorter, consisting of only a few lines of code.

1. Introduction

The constant evolution of mobile technology is moving complex AR interface techniques from desktop to hand. From the AR systems perspective, this union can bring advantages of the mobile devices' connectivity power and well known hardware interface.

The first mobile AR applications were based on laptops and translucent displays mounted on helmets adapted to receive information from various sensors. An example is presented by Feiner et al. (1997), which describes the use of AR in a mobile guide application. The bulky hardware used makes the system obtrusive to the user, difficult to maintain, and expensive.

Lately, mobile phones have become powerful enough to run AR applications. Wagner (2007) shows a set of handheld AR applications developed using the Studierstube ES library for mobile devices. Studierstube ES is a framework for AR applications development (Schmalstieg and Wagner, 2007). Mobile phones are "fully AR-capable packages" since they have all the processing power needed and hardware resources, such as inbuilt cameras and connectivity capabilities (Henrysson et al., 2005; Wagner, 2007). Henrysson et al. (2005) developed a mobile AR face to face interaction game, AR Tennis, using an optimized version of ARToolKit computer vision library (Poupyrev et al., 2002) on the Symbian platform (Figure 1).

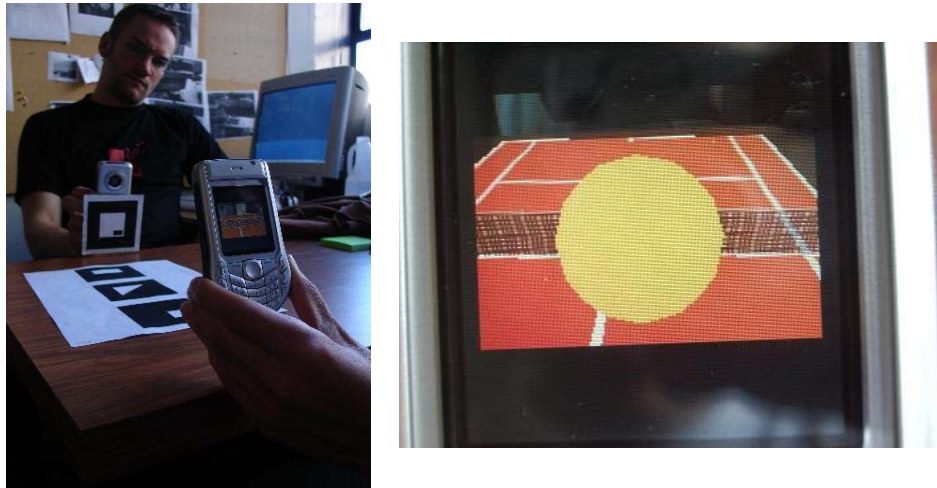


Figure 1 – Symbian AR application

An iPhone AR application can be exemplified by Paris Metro (Figure 2), a location based system that provides visual real time guide to the city of Paris, adding information to real images showing points of interest, metropolitan transport schedules, services and orientation through the city.



Figure 2 - Paris Metro for iPhones

Mobile AR applications have been deployed primarily on iPhone OS and Symbian OS based mobile devices in the last years. The importance of developing applications to these operating systems is explained by their participation in mobile market as showed in the Figure 3.

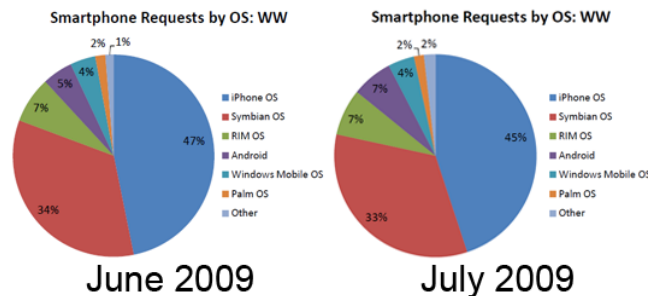


Figure 3 - Statistics of Mobile Operating System Participation in Mobile Market (AdMob, 2009)

According to recent metrics reported by AdMob Mobile Metrics Report (2009), iPhone and Symbian OS based mobile devices are the most requested smartphones across the world. Tendencies point to Android participation increasing, decreasing iPhone's share while Symbian has a relative stable participation, according to Juniper Research (2009). In addition, the same research indicates the continuous and rapid increase in mobile device sales, from 106 million units this year, to 223 million in 2014.

Despite the importance of mobile devices in the world's technology scenario, developing AR applications to such hardware is still extremely complex and time consuming. For example, because of security policies, the access to get video stream from iPhones' cameras in real time, which is a fundamental resource for AR applications, is not allowed for general applications. Symbian C++, the native programming language of Symbian OS, has a particular structure of classes, string descriptors and functions to address memory management that increase significantly the effort and time needed at development. On the other side, it unleashes efficiently most of the hardware features, including camera access and graphic processing.

Python programming language is an alternative approach to address the complex development issue. Characteristics which make Python an easy to learn language suitable for rapid prototyping include:

- Syntax similar to C/C++;
- High-level interpreted language with simple structure;
- Extensible in C/C++;
- Supports object orienting programming, namespaces, and multiple inheritance among other features typical of versatile languages.
- There are plenty of modules available for developers covering access to media resources, connectivity, mathematical libraries, etc, simplifying development.

Python has been successfully used to speed up and simplify Virtual Reality (VR) and AR prototyping processes in desktop solutions (Cooper et al., 2000; Seichter et al., 2008; Sandor and Klinker, 2004). Recently Python has been ported to Symbian OS by Nokia, the Python for S60 platform (PyS60). This version can be extended by Symbian C++, which includes access to native S60 features, and runs in S60 SDKs' emulators. PyS60 can be similarly used to simplify development of Symbian based AR applications.

AR applications in mobile devices are becoming more and more popular as the hardware power increases. However, the software development is still very time consuming because of the inherent complexity of such applications and limitations of mobile systems development platforms. This work presents Magnet, an approach using PyS60 extended in Symbian C++ to improve and facilitate development of AR applications on one of the most powerful platforms, Symbian OS based devices.

2. Basics

AR applications are characterized by interaction with digital data annotated in the real world in real time. There are many techniques to register information into real images, but particularly with mobile phones the process involves capturing video streaming from integrated cameras to later be mixed with the augmentation and be presented on the display. In the augmentation process, markers or natural features are identified by image-processing software that determines relative pose estimation for the camera and virtual objects. With the position known, the virtual camera is set and data is drawn into the live video stream.

Studierstube Tracker (StbTracker), computer vision component of the software framework Studierstube ES presented by Schmalstieg and Wagner (2007) and Wagner (2009), was developed to support handheld augmented reality is the image-processing software used in this work. It features high-performance tracking, portability and has a modular structure, facilitating its configuration. StbTracker can be configured to track different types of markers like split markers, dot markers and frame markers, shown in the Figure 4.



Figure 4 – Split marker, dot marker and frame marker (from left to right)

StbTrackers' method tracks markers by thresholding the image, searching for rectangles that indicate existence of markers, decoding contained barcode, and finally, estimating camera pose from a homography.

In this work, OpenGL ES (Khronos Group, 2009), a mobile version of OpenGL is used to draw virtual objects into the image.

3. Extending Python with Symbian C++

Symbian C++ was created to be robust, but its specific complex structures and lack of automatic memory management - task left to developers - make it very time and effort consuming. Python, on the other hand, is an open source, high-level interpreted programming language that has simple structures. PyS60 can be extended with features not covered by its standard libraries, having the same access level to resources as Symbian C++ applications have.

Productivity is improved in an extended Python development scenario by concentrating the developers' focus on application logic and user interface instead of overloading them with low level computing details. Those details are encapsulated in modules that are used as "black boxes".

The Python API defines a set of constructs to give access to Python run-time system and can be imported into a C application through the header file "Python.h". New built-in types and functions written in C can be defined and interfaced by defining

wrapper structures, and then used from Python scripts. Initially a static structure containing a list of functions that will be available to Python scripts must be written. The structure specifies a Python method name, the corresponding C function, parameter flags, and description of the function. An example can be seen in the code below:

```
static const PyMethodDef all_methods[] = {
// maps python function name to C++ function name
{"InitCamera", (PyCFunction)InitCamera, METH_VARARGS, "init
camera"},
{"TrackerInit", (PyCFunction)Init_Tracker, METH_VARARGS,
"init tracker"},
...
{0, 0}};
```

For each entry, a wrapper function must be implemented. For example:

```
static PyObject* InitCamera(PyObject* /*self*/, PyObject* /*args*/)
{ TInt error;
TRAP(error, DoInitCamera());
if (error) return SPyErr_SetFromSymbianOSError(error);
return Py_BuildValue("i", 1);
}
```

In the code above, the actual function to initialize a camera is called as a parameter of a “TRAP” instruction. The “TRAP” instruction catches leaves (Symbian equivalent to C++ exceptions), acting like a “try/catch” exception handler, since Python code won’t be able to handle leaves. “TRAP” presents errors as a Python exception.

A special case of wrapper function is a wrapper for a callback function. The use of callbacks is especially interesting to this work since it relies on a loop of image capture, image process and image drawing. The Magnet module takes care of the first two tasks and must call a Python callback function to complete every loop. The wrapper in Magnet for the camera callback function is defined as follows:

```
static PyObject * camera_callback(PyObject *dummy, PyObject *args)
{
PyObject *result = NULL;
PyObject *temp;

if (PyArg_ParseTuple(args, "O:set_callback", &temp)) {
if (!PyCallable_Check(temp)) {
PyErr_SetString(PyExc_TypeError, "must be callable");
return NULL;
}
Py_XINCRREF(temp); // Add a reference to new callback
Py_XDECREF(my_callback); // Dispose of previous callback
my_callback = temp;
Py_INCREF(Py_None);
result = Py_None;
}
return result;
}
```

In this function the method “PyArg_ParseTuple” parses the argument object and associates a Python method to the object “my_callback”. Every time the image capturing and image processing is completed, the function “get_CameraCallback” is called. This results in a call to the Python callback function using the “PyEval_CallObject” method:

```
static void get_CameraCallback(){
    PyObject *arglist=NULL;
    PyObject *result=NULL;
    int arg = 1;
    // build arguments
    arglist = Py_BuildValue("(i)", arg);
    Py_INCREF(my_callback);
    // Calling Python function
    result = PyEval_CallObject(my_callback, arglist);
    Py_DECREF(arglist);
    Py_DECREF(my_callback);
    if (result == NULL) return; // Pass error back
    Py_DECREF(result);
}
```

Parameters and return values passed to Python functions are constructed using the “Py_BuildValue” method which casts C++ types to Python objects. More details on extending Python in C++ is shown in *Symbian beta* (2009), by van Rossum and Drake (2009) and Nokia (2009).

4. Magnet

The Magnet extension is a set of classes to access camera and STBTracker frame marker detection features implemented in native Symbian C++ and wrapped to be used by Python scripts. It includes:

- Camera access
- OpenGL ES initialization
- Background image rendering
- Detection of frame markers

It is responsible for camera initialization, image processing (e.g. marker tracking), and drawing the real image as background on the OpenGL ES environment. These tasks, excluding initialization, are repeated in an infinite loop, and at the end of each loop a callback Python function is expected to introduce the augmentation – a 3D model or any other data/interaction designed for the application.

The Magnet module makes available a small and simple set of functions, described as follows:

- CreateCamera(): create instance of camera and tracker manager.
- TrackerInit(): initialize tracker – the tracker detection is initialized to detect any frame marker and return it’s code.

- `InitCamera()`: initialize camera/marker detection loop. At the end of every loop, the background image is rendered and a callback function is called.
- `InitGLES()`: initialize GLES to render background image and any 3D model needed.
- `GetP()`: gets P matrix – returns a tuple with 16 values that corresponds to the pose estimation 4x4 matrix.
- `GetT()`: gets T matrix – returns a tuple with 16 values that corresponds to the transformation 4x4 matrix.
- `SetCameraCallback(aFunction)`: set camera callback function to “aFunction” previously defined in Python.

Magnet supports S60 3rd edition phones. It is installed as a “pyd”, a dynamically loaded library file in the “c:/sys/bin” directory of the mobile device.

The library must be imported at the beginning of the script as follows:

```
if e32.s60_version_info>=(3,0):
    import imp
    magnet=imp.load_dynamic('Magnet', 'c:\\sys\\bin\\Magnet.pyd')
else:
    import Magnet

from Magnet import *
```

Once imported, it is necessary to create a Python function to be called at the end of image capturing/marker detection loop. The callback function must be responsible for processing data acquired from markers, including marker code, pose estimation and transformation matrixes. The marker code is returned as an integer, where “-1” indicates no marker detected. The matrixes are 4x4 OpenGL ES formatted data and can be accessed by functions “GetP()” and “GetT()”.

In the first test example, a callback function named “frameback” with mostly OpenGL code was written to draw a 3D model previously loaded from an “.obj” file. The function must be registered using “SetCamera Callback(aFunction)”.

Inside the application’s main method, the order of functions to properly initialize the image capture/marker detection loop is:

1. `CreateCamera()`
2. `InitGLES()`
3. `TrackerInit()`
4. `InitCamera()`

Before the 4th step, eventual 3D models or any other media necessary to augment the video streaming should be loaded, to optimize the loop that starts with the next function.

Magnet is still in early stages of development but results in development process improvement can already be noticed. A simple “hello world” AR style, e.g. a single marker detection with augmentation of a 3D model, was built with less than 100 lines of code. This is a significant result since the same application in Symbian C++

language demanded 7 classes and nearly 1500 lines of code. Figure 5 presents the application.

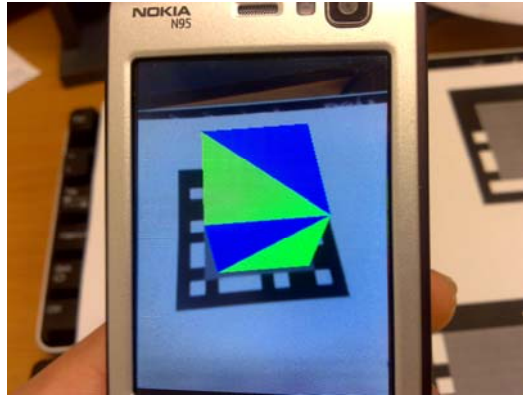


Figure 5 – Magnet test

A third party open source 3D model loader was used in the test. A benchmark test indicated an average rate of 11 frames per second when drawing 3D objects on top of a single marker, tested with different 3D objects in N95 and N96 models of Nokia phones.

5. Discussion and Future Work

The constant increasing power of mobile phone devices is enabling them to support AR applications. However, the complex nature of these applications, characterized by real time interaction with digital data registered in the real world, and limitations imposed by mobile device platforms make mobile AR systems development very effort demanding and time consuming. To simplify such tasks in one of the most popular mobile device platforms, Symbian OS based devices, Magnet presents a hybrid language development approach.

The concept behind Magnet's development is that by encapsulating low-level computing functions, developers can focus on the application logic and user interfaces, thus improving productivity. This goal is sought by creating a wrapper to camera and image processing, tasks that are efficiently implemented in Symbian C++, to be easily integrated to Python, a high level language. Magnet uses StbTracker computer vision library to detect markers and OpenGL ES to draw images.

The implementation so far allows developers to start a loop of video streaming with frame marker detection and add a callback function at the end of each loop to augment the video streaming. The results showed a significant reduction in lines of coded needed in comparison to pure native AR applications and acceptable performance.

As future work, Magnet will be improved by:

- Including functions to change calibration of camera, in order to make it portable to different mobile devices;
- Enabling use of multiple marker detection, simultaneously;
- Enabling detection of natural features, such as head and hands tracking;

The goal is to turn Magnet into a flexible and fully customizable module to support augmented reality applications' basic tasks, for example video streaming capturing and image processing. With Magnet integrated into development and rapid prototyping processes, it is expected that they will become quicker and easier, improving the quality of future mobile augmented reality systems.

6. References

AdMob. 2009. AdMob Mobile Metrics. [Online]. Available: <http://metrics.admob.com/2009/08/july-2009-metrics-report/> [7 October 2009]

Cooper S., Dann W., Pausch R. 2000. Alice: A 3-D Tool for Introductory Programming Concepts. *Journal of Computing Sciences in Colleges*, 15, 5 107-116.

Feiner, S., MacIntyre, B. e Höllerer, T.. 1997. A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment. *Proceedings of First International Symposium on Wearable Computers*, 74-81.

Henrysson, A., Billinghurst, M., and Ollila, M. 2005. Face to Face Collaborative AR on Mobile Phones. *In Proceedings of the 4th IEEE/ACM international Symposium on Mixed and Augmented Reality. Symposium on Mixed and Augmented Reality*. IEEE Computer Society, Washington, DC, 80-89.

Juniper Research. 2009. Juniper Research. [Online]. Available: <http://www.juniperresearch.com/shop/viewreport.php?id=191> [7 October 2009]

Khronos Group. *OpenGL ES*. 2009. [Online] Available: <http://www.khronos.org/opengles/> [7 October 2009]

Nokia. 2009. *Forum.Nokia.Com Getting Started with Python for Series 60 Platform*. [Online] Available: <http://www.forum.nokia.com/> [7 October 2009]

Poupyrev I., Kato H., Billinghurst M. Artoolkit. http://www.hitl.washington.edu/research/shared_space/download/

Sandor C., Klinker C.. 2004. A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality. *In Journal for Personal and Ubiquitous Computing*. Springer Verlag.

Seichter, H., Looser, J., Billinghurst, M. 2008. ComposAR: An Intuitive Tool for Authoring AR Applications. *In: Saito, M.A.L., Oliver, B. (eds.) International*

Symposium of Mixed and Augmented Reality (ISMAR 2008), pp. 177–178. IEEE, Cambridge.

Schmalstieg D., Wagner D., 2007. Experiences with Handheld Augmented Reality. *Proc. IEEE International Symp. on Mixed and Augmented Reality*.

Symbian beta. 2009. Symbian beta. [Online] Available: http://developer.symbian.org/wiki/index.php/Multi-Language_Programming_-_Combining_Python_with_Flash_Lite_and_Symbian_C%2B%2B [7 October 2009]

van Rossum G., Drake Jr. F. L., 2009. Extending and embedding the python interpreter [Online] Available: <http://docs.python.org/ext/ext.html>. [7 October 2009]

Wagner, D. 2007 Handheld Augmented Reality, PhD dissertation, Graz University of Technology.

Wagner D. *Handheld Augmented Reality*. 2009. [Online] Available: http://studierstube.icg.tu-graz.ac.at/handheld_ar/stbtracker.php [7 October 2009]