Android Controller

Jeanne Wang

Abstract

Taiwan's National Center for High-Performance Computing (NCHC) has developed large title display walls, capable of both 2D and 3D display. Also, the California Institute for Telecommunications and Information Technology (Calit2) has CAVE Virtual Reality environments. These large displays make possible new immersive applications. To take full advantage of such a system, a dynamic controller must be used so a user can firmly engage with the space. Given the proliferation of new sensor enabled, network connected, portable devices, this project aims to explore novel controller applications using such a device. A tablet was chosen specifically for its large form factor and high computing power.

Introduction

NCHC has a large dataset involving the Morakat Typhoon and its aftermath. We thought a simple application would be to display a "before" image of an area in Kaohsiung, Taiwan on a large screen and to display a window of the "after" image on the tablet. A user would hopefully see a seamless transition between "before" and "after" as he pans. This project uses the Acer Iconia Tab A500 a 10.1 inch tablet running Android 3.0 (Honeycomb) and comes equipped with a gyroscope, accelerometer,compass and rear/front facing cameras. Also the Iconia boasts a Dual-core 1GHz ARM Cortex-A9 proccessor, ULP GeForce GPU and Tegra 2 T20 chipset [1]. While I did play with all of the sensors, I only used the accelerometer and camera for most of the application. Future work would include combining different sensor data

for more robust findings.

Implementation Details

ARToolkit is a C software library for augmented reality functionality [2]. Its key functions include tracking a users viewpoint using markers via computer vision techniques. In this project, I used AndAR, a Java/Android port of ARToolkit [3]. AndAR was used on the client side (tablet side) of the application whereas the server side ran Collaborate Visualization and Simulation Environment (COVISE), a distributed visualization software [4]. Most of my COVISE plugin was actually written using the scene graph provided by OpenSceneGraph [5].

Camera Calibration

For accurate detection of the marker and then for accurate transformation matrix readings, ARToolkit has to be calibrated for the specific camera being used. Camera calibration generally involves finding correct intrinsic camera properties: focal length, scale/skew factors, image center, and lens distortion [6]. ARToolkit does come with a calibration utitlity program but it directly interacts with a web camera. As I needed to calibrate my tablet's camera which could not directly interface with the utility program, I modified ARToolkit's calib_camera2 utility to take images from the file system instead of from a video stream. This modification involves turning off the video stream, grabbing still images from the file system, decompressing these images (in this case jpeg) into ARToolkit's ARUint8* (BGR format, 3 bytes per pixel) and from there passing into existing calibration functions. While this was great for an initial calibration, the Iconia Tab has a mechanical auto-focus which changes the focal length. I used a simple one time-calibration so my measurements were off due to the auto-focus. Building in a self

calibration into this application would make measurements much more accurate.

Android Application

AndAR ports the C library ARToolkit by using the Java Native Interface (JNI); all of the key functionality of ARToolkit and many of the utilities have been ported over. Using the library, I was able to get the transformation matrix between the camera and the marker. This 3x4 transformation matrix is of the form [xorientation, yorientation, zorientation, translation] where each of these have an x,y,and z component. Again, these measurements are off due to my single camera calibration. As this tablet also has a gyroscope, it would be nice to compare orientation between the two. Note that ARToolkit finds orientation of a marker which may not be exactly at the same orientation as the tablet's normal projection onto the plane. Using this information, it is possible to track a person if the marker is in view. But what if the marker is not in view? Using accelerometer data, we can roughly calculate the movement of the tablet. We approximate using uniform acceleration linear motion equation $s=s_i+v_it+1/2at^2$. Obviously a user will not behave according to uniform acceleration except in very small bursts. We wait until the initial velocity is zero before taking a time sample, and then sample very quickly every time the sensor throws a onSensorChanged event. Since we can move the marker along with the tablet, we can keep the marker continuously in view. The client sent out UDP packets containing movement information.

My application takes advantage of OpenGL ES, an embedded systems graphics library [7]. The main use is creating a simple model from a 2D image. This image is simply applied as a texture to a two dimensional quadrilateral which is then rendered as two separate triangles. I created such a model with the Kaoshiung "after" image which

was then registered with the Hiro marker pattern. Note that much more sophisticated models are possible to display, this just happened to be the data I was using.
While moving in and out in the z axis does change the model viewing size, I wanted to really be able to zoom in and out of features. By watching the accelerometer in the z axis, we can figure out when a user has moved to a new position in the z axis. From there we can get the distance from the screen using the translation vector, and then recreate our model's texture to scale. Recreating our texture and binding it is actually quite computationally expensive for a large graphic (in this case 1300x900) and so is a bit too slow for real time.

COVISE Plugin

While COVISE is freely available for academic users, I had quite a few issues getting it to install correctly on my machine running Ubuntu 11(Natty Narwhal). For others trying to install COVISE, the main errors I encountered were dynamic linking issues with COVISE libraries such as libcoVRconfig.so.6, which can be fixed relatively easily with the command line tool ldconfig. Also the correct version of OpenSceneGraph must be installed for that particular version of COVISE. Also, be sure to set the COVISE specific environment variables such as COVISE_PATH, COVISEDIR, and COVISEDESTDIR.
Plugins for COVISE are written in C++ and can use any C,C++ libraries you wish to include. While I could have used ARToolkit as a library for COVISE, I actually did all of my ARToolkit computation on the tablet itself. In fact the standard AndAR application even runs on the original Google G1 which says something about the minimal processing power required to run ARToolkit even in realtime.

The plugin was basic, consisting of a server aspect that waited for UDP packets from the client telling it new positions for the marker. These UDP packets contained JSON encoded arrays representing the acceleration data and the transformation matrix. Then the plugin would redraw its surface with an updated marker position on top of the Kaoshiung “before” image. The plugin was multi-threaded with one thread that listened for packets and the other doing the drawing.

Conclusions

A tablet can be a great tool for controlling applications due to its portability and multiple input sensors. It can be very useful for creating interactive displays. In this case, I created the application so that I could use a tablet but I think in general, the use case should be the opposite. A controller should be created for the application, and as such, use tablet sensors and inputs as necessary.

My application was quite jittery in its implementation and was never actually tested on a large screen. I believe that on a multi-tiled screen, using a marker based technique would run into issues with the screen bezels, as bezels would split marker display resulting in poor detection. Also the bezel around the screen on the tablet hindered a seamless experience.

A big issue I ran into was measurement inaccuracy. ARToolkit gives its measurements based on its intrinsic parameters and a known size of marker. This is great for physical markers that you print out onto paper, but in the case of displaying it on a screen, screen size and dots per inch (DPI) come into play. So all my measurements were off when displaying it on a new screen (especially bad on a projector which can dynamically change size) in addition to the calibration error. Also for a projector, many

were not bright enough for a good image that ARToolkit could process. Back-lit LED screens fared much better.

Future Work

While a tablet based approach to controlling large displays would be useful, it should integrate multiple sensors for more stable data. Using ARToolkit, my application was quite shaky as it was constantly looking for the marker, maybe using acceleration change would have been a good indicator to ARToolkit to search for a marker. If I could have integrated the gyroscope, I could have used it's data to strengthen my orientation data. Future work should look into Kalman filters for both smoothing out jittery data and also for sensor fusion. Also taking into account not only where the tablet is in terms of the marker, but where the user is in terms of the tablet would be useful for creating an optimal view for the user. Future work should also look into creating a 3D implementation.

Acknowledgements

Thank you to Jurgen Schulze and Fang-Pang Lin for being my mentors for this project. Another thank you to the members of NCHC for being very welcoming hosts. And a third thank you to Pacific Rim Undergraduate Experiences (PRIME) who made this project possible.

References

[1] *Acer Iconia Tab Specifications*
http://www.gsmarena.com/acer_iconia_tab_a500-3907.php

[2] *ARToolkit*
http://www.hitl.washington.edu/artoolkit/

[3] Domhan, Tobias. *AndAR*
http://code.google.com/p/andar/

[4] *COVISE*
http://www.hlrs.de/organization/av/vis/covise

[5] *OpenSceneGraph*
http://www.openscenegraph.org/projects/osg

[6] Duraiswami, Ramani. *Camera Calibration*
http://www.umiacs.umd.edu/~ramani/cmsc828d/lecture9.pdf

[7] *OpenGL ES*
http://www.khronos.org/opengles/