Dance With Your Hands

ESE 350 Final Project
Spring 2011-05-09
Philip Peng and Eric Chen
University of Pennsylvania

http://dancewithyourhands.blogspot.com/
http://beatsportable.com/2011/05/beats-kinect-win/
http://beatsportable.com/static/kinect/
http://www.seas.upenn.edu/~ese350/

Table of Content

1. Introduction

The original motivation for our final project was to build something interesting related to music using a Kinect and a Beagleboard. The ideas we came up with included implementing a Theremin-like instrument or building a custom “air instrument”. During our project proposal, feedback from the TAs prompted us to consider simulating the conducting of a music piece. After toying around with several ideas, we realized that players are essentially the conductor in a music piece in music rhythm games, and that conducting resembles dancing with one’s hands.

Music rhythm games have found great audience in the video game industry, especially since the popular reception of the Dance Dance Revolution (DDR) franchise. Its core gameplay involves the player stepping his or her feet on four directional buttons to the general rhythm of a song. Due to this success, many other games with similar gameplay such as tapping buttons with hands have been created. Many more recent innovative games such as Guitar Hero did explore new grounds with the simulation of playing various instruments. Still, this video game genre has never moved away from relying physical buttons for inputs.

As a consequence of these limiting input devices, players are restricted from being able to actually dance with their hands in the air. With the power of the Microsoft Kinect, we wanted the game that popularized this video game genre, DDR, to live up to its name to not only allow players to dance with their feet but also their hands. The goal of our final project is to eliminate the physical input device in a music rhythm game to replace the input device with a Kinect.

This final project makes use of a Microsoft Kinect running on a Beagleboard to detect player’s hand movements. The inputs are used by the mobile game Beats (developed by Philip previously) running on Android 2.2. We chose Android OS because of its faster performance over Ubuntu and because Beats runs out of the box. We aimed to develop a reliable system for recognizing player’s hand in the space in front of their chest while maintaining the game’s playability.

2. Hardware

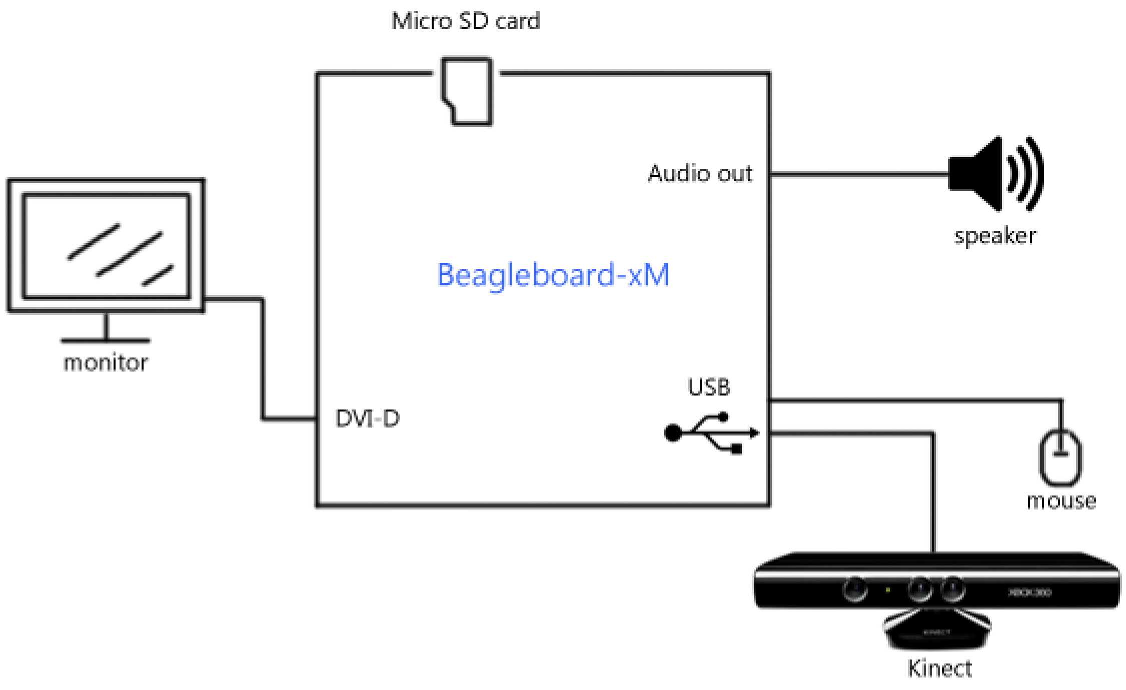
a) Hardware List

- 1x Microsoft Kinect + USB cord + power cable
- 1x Beagleboard-xM + power adaptor
- 1x 4+ GB micro SD card
- 1x Display monitor + HDMI cable
- 1x Speaker

b) Hardware Description

The Kinect communicates with the Beagleboard via one of the onboard USB ports. The Android OS is installed onto the micro SD card on a separate computer and inserted into the micro SD slot. Beagleboard then outputs the video to our display through its DVI-D port. The audio outputs through the 3.5 mm audio output jack to our speakers.

c) Hardware Diagram



4. Plans

a) Angstrom vs Ubuntu vs Android

The Beagleboard provided three options in terms of which operating system to work with, Angstrom, Ubuntu, and Android. We will investigate each of these further and decide which operating system would be optimal for our purposes.

b) StepMania Simulator vs Beats

We will also have to decide which music rhythm game will be used for this project, specifically between StepMania and Beats, depending on which operating system we choose. StepMania runs on Ubuntu while Beats was developed for the Android platform. The main factor of consideration is the performance of the game on Beagleboard. While StepMania+Ubuntu provides more functionalities and is more fully featured, Beats+Android is likely to result with better performance.

c) Beagleboard vs USB device

Another important design decision to be made is whether or not we will run the game directly on the Beagleboard. If we decided against it, the Beagleboard will serve as an interface device that processes Kinect data and sends fake keystrokes to a separate computer running the game. Again, this choice will depend on the outcome of several speed tests on the Beagleboard.

5. Experiments

a) Android + Beagleboard
Our very first step was to get our Beagleboard up and running. Angstrom booted without problems, but we chose Android for our first trial. Following the instructions found here, we prepared our SD card with Android 2.2 and booted it on our Beagleboard successfully. However, this success is short lived, followed by our Beagleboard overheating from unknown reason.

b) Kinect + Fedora
With our Beagleboard temporarily disabled, we proceeded to explore Kinect, running it on Fedora. The setup process was straightforward and we experimented with a few demos. One demo, glview, was modified to allow the Kinect to output the corresponding area being detected. The text outputs are top-left, top-right, bottom-left, bottom-right, and middle ("TL", "TR", "BL", "BR", "MM").

c) Ubuntu + Beagleboard + StepMania
After obtaining a new Beagleboard, we decided to test out Ubuntu and compare the performance against that of Android. We also installed StepMania simulator, only to be disappointed by the low frame rate at which the game ran.

d) Android + Beagleboard 2
Since Ubuntu was eliminated from our choices of operating system due to its lacking of speed, we reverted back to Android 2.2. This time, without the Beagleboard overheating, we installed Beats and are surprised to find that it runs at the full framerate. This allowed us to proceed with the next step.

e) Kinect + Android
The next major milestone was getting Kinect to play well with Android. Unlike running on Fedora or Ubuntu, configuring Android to accept Kinect was a bit tricky. However, after it was setup, the demo codes were modified in the similar way as before.

f) Beats + Kinect + Android
Finally, we were ready to combine each component together. Philip modified his Beats source code, integrating the functionalities from the modified demo, allowing inputs to be detected from the Kinect and entered in the Beats game.

6. Final Result

a) Video

The following demo video was taken on May 5th during the ESE350 final projects demo day in the Denkin Lab, Moore Building, UPenn. The video is currently available for viewing on YouTube but is also included in the `video/` folder. The unedited video was captured by Phil's Samsung Captivate at 1280x720, then directly uploaded to YouTube.

Beats, Advanced Rhythm Game - 1.5.5b Kinect PoC Demo [Android]
http://www.youtube.com/watch?v=o1MsL-VbyPg
Full video description:

> http://beatsportable.com/
> "Beats, Advanced Rhythm Game" is a music rhythm game for Google Android phones/devices. Dance with arrows to the beat of music StepMania/DDR (Dance Dance Revolution) style or tap circles Osu! Tatakae! Ouendan/ Elite Beat Agents style!
>
> This is a proof-of-concept gameplay demo of Beats 1.5.5b with Microsoft XBOX Kinect support. The song used in this demo is "Chaoz Impact" by ParagonX9, stepfile by Lisek with permission for distribution with Beats. The person learning to "dance with his hands" in the demo is my lab partner, Eric Chen while I (Philip Peng) am recording the video. This was our final project for ESE350 (Microcontrollers & Embedded Systems) at University of Pennsylvania http://www.seas.upenn.edu/~ese350/
>
> The Kinect is connected to a Beagleboard-xM running Android (Froyo 2.2 via the Rowboat project), which is running a modified version of Beats that includes Kinect support via OpenFramework. For details, see the forum post at http://beatsportable.com/forum/viewtopic.php?f=4&t=1023 (includes a link to our blog with detailed steps on setting things up as well as the modified source code and apk for download).
>
> For the latest official release (aka the normal phone version), download this app via the Android Market (search for "beats", "rhythm", or "stepmania") or visit http://beatsportable.com/downloads/
>
> For more information, see http://beatsportable.com and leave your feedback on the forums at http://beatsportable.com/forum/

b) Install Instructions

Here are the over-verbose, text-only installation steps for installing and running the modified Beats with kinect support on your Beagleboard-xM. These steps assume that you have all the necessary hardware and downloaded files in the `install/` folder.

1. Hardware checklist:
    a. Beagleboard-xM with power cable
    b. microSD card (4GB minimum) with adapter (if needed)
    c. XBOX 360 Kinect with power cable and USB adapter
    d. TV/monitor with some cable combination allowing HDMI input
    e. USB mouse and USB keyboard
    f. Computer running Linux and with an SD card reader/adapter
2. Install Android on Beagleboard:
    Original files and instructions can be found at http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/TI_Android_DevKit/02_00_00/index_FDS.html
    a. Place microSD card into adapter (if necessary) and connect it to your laptop running Linux (our laptop was an Asus Eee PC running Ubuntu 10.04 Server Edition). Note that if you are not logged in as root (`sudo -su`), you will have to append `sudo` to all your commands.
    b. Extract the tarball of the pre-build Android image:
    ```
    tar -xzvf beagleboard-xm.tar.gz
    cd beagleboard-xm
    ```
    c. Find the device letter of your inserted SD card - look for a device represented by `/dev/sdX` where *X* is a lowercase letter (e.g. b, c, d, etc) and with the right storage size. Be careful: `/dev/sda` is usually your computer's hard drive.
    ```
    fdisk -l
    ```
    Also check to make sure that is not on the list of mounted devices via:
    ```
    mount
    ```
    If it is mounted at, for example, /mnt/sdcard, unmount it via:
    ```
    umount /mnt/sdcard
    ```
    d. Prepare the SD card via the provided script with *X* being your device letter:
    ```
    ./mkmmc-android /dev/sdX
    sync
    cd ..
    ```
3. Make sure Android boots up properly on the Beagleboard:
    a. Insert microSD card into Beagleboard's microSD slot.
    b. Connect USB mouse, keyboard and HDMI cables to Beagleboard.

   c. Plug in the power adapter to power on Beagleboard.
   d. Wait for Android to boot up (first time can take ~4 minutes or so).
   e. See http://processors.wiki.ti.com/index.php/TI-Android-FroYo-DevKit-V2_UserGuide#Keypad_mappings for keyboard mappings and more user guide info.
4. Install modified kernel with high-speed USB support (needed for Kinect):
   Modified kernel was built using instructions found at http://www.noritsuna.com/archives/2011/01/openframeworks_kinect_android.html
   a. Unplug Beagleboard's power plug to power off Beagleboard.
   b. Remove the microSD card and plug it back into your computer.
   c. This time you will notice that your SD card has three partitions. Make three mount points for these three partitions and mount them:
```
fdisk -l
mkdir /mnt/android-boot
mkdir /mnt/android-rootfs
mkdir /mnt/android-data
mount /dev/sdX1 /mnt/android-boot
mount /dev/sdX2 /mnt/android-fs
mount /dev/sdX3 /mnt/android-data
```
   d. Copy over the modified kernel, overwritting the old one:
```
mv /mnt/android-boot/uImage /mnt/android-boot/uImage-old
cp uImage /mnt/android-boot/
```
5. Modify the init.rc file to enable USB support on startup:
   a. Add in the line `mount usbfs none /proc/bus/usb -o devmode=0666` to `init.rc` in the rootfs partition, somewhere after the initiation lines and before the `on boot` lines. You may use the pre-modified init.rc file:
```
mv /mnt/android-fs/init.rc /mnt/android-fs/init.rc-old
cp init.rc /mnt/android-fs/
```
6. Copy over the .apks for installation and Beats data files:
   a. Copy over the apk for the modified Beats:
```
cp beats-r475-kinect.apk /mnt/android-data/
```
   b. Copy over the apk for the modified ofsample and aLogcat if you wish:
```
cp ofsample-mod-unsigned.apk /mnt/android-data/
cp alogcat-2.3.apk /mnt/android-data/
```
   c. Copy over sample song data files for Beats:
```
unzip samples.zip
mkdir /mnt/android-data/Beats
cp -r Songs /mnt/android-data/Beats/
```
7. Boot up Android and install the apps
   a. Unmount SD card:
```
sync
umount /mnt/android-boot
umount /mnt/android-fs
umount /mnt/android-data
```

b. Attach the Kinect to the Beagleboard and boot up Android, then open the web browser.
c. Ignore the “No internet connectivity” warning and type the following in the web browser’s address bar:

    `file:///sdcard//beats-r475-kinect.apk`

d. Click “Go” and then click “Next”, etc. to install the Beats. If you have any troubles installing the app, you may need to check “`Unknown Sources`” under “`Menu -> Settings -> Applications -> Development`” (middle click your mouse or press F1 on your keyboard to open the Settings panel).
e. Click “Close” after installing Beats. Repeat previous step for the ofsample `ofsample-mod-unsigned.apk` and aLogcat `alogcat-2.3.apk`

8. Test the installed apps:
    a. Run any of the installed apps by clicking on the tab icon on the home screen and selecting the appropriate app icon.
    b. The ofsample app will help you test out and position yourself relative to the Kinect. The numbers at the bottom will help you calibrate the “`Kinect Distance`” setting in Beats, which is actually a misnomer (see code explanation section for more details).
    c. The aLogcat application can help you read error messages and figure out what went wrong (if anything went wrong), but be warned: logcat messages are not easy to interpret.
    d. The Beats application is, of course, the modified build of the Android game “Beats, Advanced Rhythm Game” with Kinect support for standard and reverse game modes. Make sure to play around with the “`Kinect Distance`” setting under “`Tapbox Settings`” and get a feel for “dancing with your hands”.

c) Compiling Instructions

Here are the over-verbose, text-only instructions for compiling the modified kernel, modified ofsample, and modified Beats. These steps assume that you have all the necessary hardware and downloaded files in the `src/` folder. Note that, as of the time of this writing, unless you are either Philip Peng or Matt Croop, you will be unable to actually compile Beats as the program is not open source. The instructions are here nevertheless for those interested in integrating Kinect support with other Android apps or if Beats is open sourced in the future.

Modified kernel compiling instructions:
Based on http://processors.wiki.ti.com/index.php/TI-Android-FroYo-DevKit-V2_UserGuide#Kernel and http://www.noritsuna.com/archives/2011/01/openframeworks_kinect_android.html

1. Create a new Linux user named `noritsuna` and log in under that username. This is done for convenience of paths such that no modifications are needed in the pre-made development environment.
```
adduser noritsuna
passwd noritsuna
<enter a password>
su noritsuna
```
2. Extract the pre-made “Full "OpenFrameworks x kinect x Android" Development Environment for beagleboard” (modify the line so it properly points to where `openframeworks_kinect_android_fullIDE_beagleboard-20110103.tar.gz` is):
```
mkdir kinect
cd kinect
tar -xvzf <path to IDE tar.gz>
```
3. Add the pre-built toolchain path to your PATH variable (this is one line):
```
export PATH=/home/noritsuna/kinect/android-ndk-r4-crystax/
build/prebuilt/linux-x86/arm-eabi-4.4.0/bin$PATH
```
4. Extract kernel source code (modify the line so it properly points to where `Android_Linux_Kernel_2_6_32.tar.gz` is):
```
tar -xvzf <path to kernel tar.gz>
cd Android_Linux_Kernel_2_6_32
```
5. Clean and configure the kernel:
```
make CROSS_COMPILE=arm-eabi- distclean
make CROSS_COMPILE=arm-eabi- omap3_beagle_android_defconfig
```
6. Modify the kernel’s .config to include high-speed USB support:
```
echo CONFIG_USB_DEVICEFS=y >> .config
```
7. Compile the kernel. If you come across any errors complaining about lacking the command `mkimage`, install the uboot-mkimage package via your distro’s package manager (e.g. `sudo apt-get install uboot-mkimage`), then rerun the command.
```
make CROSS_COMPILE=arm-eabi- uImage
```

```
cd ..
```

8. If the cross-compiling was successful (kernel compiling takes a minimum of 10 minutes), you will have a nice kernel file `uImage` in the `arch/arm/boot` directory.

Modified ofsample compiling:

1. Apply the patch to the original ofsample source code (modify the line so it properly points to where `ofsample-mod.diff` is):
   ```
   cd android-ndk-r4-crystax/apps
   patch -p0 < <path to diff>
   ```
2. Compile the JNI code:
   ```
   cd ofsample
   ../../ndk-build -B
   ```
3. Open up eclipse
   ```
   cd ../../..
   cd elipse
   ./eclipse
   ```
4. Build and export the ofsample project as an APK via instructions on the Google Android Developer's website (or just run the project as an Android app if you do not want to sign the apk) http://developer.android.com/guide/publishing/app-signing.html

Modified Beats compiling

1. SVN checkout the Beats source code to the NDK's apps folder. Note that the modifications were done to revision 475 and may not patch cleanly in future revisions. Also note that the private SVN server that the Beats source code is hosted on does not allow for anonymous checkout, so these instructions are just educational ;)
   ```
   cd ../android-ndk-r4-crystax/apps
   svn checkout https://***/svn/beats/ beats
   ```
5. Apply the patch to revision 475 of the Beats source code (modify the line so it properly points to where `beats-r475-kinect.diff` is):
   ```
   patch -p0 < <path to diff>
   ```
6. Compile the JNI code:
   ```
   cd beats
   ../../ndk-build -B
   ```
7. Open up eclipse
   ```
   cd ../../..
   cd elipse
   ./eclipse
   ```
8. Import, build and export the Beats project as an APK via instructions on the Google Android Developer's website (or just run the project as an Android app if you do not want to sign the apk) http://developer.android.com/guide/publishing/app-signing.html

7. Development Decisions

Throughout the development of this project, we changed platforms and approaches numerous times. The three platforms/Linux distros we could have worked with were Angstrom, Ubuntu, and Android. The three libraries we could have worked with were OpenKinect's libfreenect, OpenFramework's ofxkinect and PrimeSense's own OpenNI initiative. The three possible rhythm game engines were StepMania, FlashFlashRevolution, and Beats, Advanced Rhythm Game. The three possible input->game paths/approaches were to send input via serial-port to PC running a special interpreter program, send via USB to PC and emulating a standard keyboard, or to run on Beagleboard directly. The end combination that we chose (Android + ofxkinect + Beats + Beagleboard) was a result of our experiments (described earlier) and analysis of complexity for completion (given our limited time and resources). The following is a summary of those results and decisions.

a) Platform

Angstrom:
http://beagleboard.org/project/angstrom/

Angstrom is the default Linux distribution that comes with the Beagleboard. It is very small, lightweight, and supposedly fast and well optimized for the board. Unfortunately, due to its small user base and support community, there does not exist any extensive repository for software packages, meaning all software and development libraries would have to be compiled from scratch.

Considering the Beagleboard's limited processing power and our own time constraints, we decided against sticking with Angstrom as much of our time would probably be spent compiling code instead of writing it. Proceeding with Angstrom could be possible - we just decided that it was not time-feasible.

Ubuntu:
http://elinux.org/BeagleBoardUbuntu
http://tirokartblog.wordpress.com/2011/01/22/kinect-on-the-beagleboard-xm-not-working-yet/

As the current most popular Linux distro among consumers, Ubuntu has a very large user base and support community. The netbook version of Ubuntu was fully complete and operational on the Beagleboard. Most, if not all, of the packages we looked for throughout the project had prebuilt ARM binaries available through the main repositories, so software installation was very easy and convenient. With all the features of a full-fledged desktop, it was definitely the preferred platform to work on.

Excess overhead wouldn't be an issue either as feature removal is always easier than feature implementation. Our plan with Ubuntu was first to get things working (with the extra services and desktop overheads), then re-install everything with the slimmed down server version. In addition, even with all the overhead, Kinect data processing stayed at a good and steady 20 FPS (frames per second). Eventually we chose Android over Ubuntu due to the poor performance of StepMania on Ubuntu - see below.

Android:
http://code.google.com/p/rowboat/
http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/TI_Android_DevKit/02_02_00/index_FDS.html

Google's Android OS uses the Linux kernel but runs Java software through a custom Java VM called the Dalvik Virtual Machine. Although it is lightweight in terms of features, the fact that software is run on a virtual machine as opposed to natively means that it is innately slower than a real Linux distro (such as Ubuntu). The version that we used was also outdated and incomplete hardware support (Froyo 2.2 instead of the latest Gingerbread 2.3).

While Android was not the ideal platform, it was capable of running a DDR-type game at very high framerates. Both StapMania (a third party StepMania port) and Beats (Philip's StepMania clone) ran nicely at 50+ FPS. With the extra overhead, data processing for the Kinect was much more limited and we were only able to obtain 15-20 FPS. The final decision to use Android, however, was because 1) of its ability to run Beats smoothly and 2) it sounded more interesting; we found very few Kinect x Android projects posted online.

b) Library

libfreenect:
http://openkinect.org/wiki/Getting_Started
https://github.com/OpenKinect/libfreenect

libfreenect is a simple, barebone library+driver used in the OpenKinect project. It simply interprets data sent from the Kinect (thus its only dependency is libusb) and gives you a nice API for accessing the raw data. Philip was able to run and manipulate the data very easily on both Fedora (on his laptop) and Ubuntu (on Beagleboard) at a smooth 60 FPS and 20 FPS respectively. Unfortunately, there were no available documentation or instructions on adding libfreenect support in Android. We could have tried playing around with making our own Java wrapper for the library, but we decided to only default to that if other options failed.

OpenNI:
http://www.openni.org/
http://tirokartblog.wordpress.com/2011/01/21/kinect-working-on-ubuntu-10-10/

The OpenNI drivers come from PrimeSense, the company behind the actual Kinect technology. Their website actually only went public recently (end of last year) so there isn't a long history of hacking with their drivers and provided software. While the binary demos ran perfectly fine on Linux and Windows, compiling from source code turned out to be rather tricky, especially as the latest source code available on GIT had compatibility issues and the released unstable source code tarballs actually lacked Kinect support. Compiling the code on Ubuntu on the Beagleboard was worse due to lack of pre-compiled libraries for ARM. Eventually we just gave up on working with OpenNI out of lack of documentation (and subsequently, time).

ofxkinect:
http://www.openframeworks.cc/
http://www.noritsuna.com/archives/2011/01/openframeworks_kinect_android.html
ofxkinect is one of the many plugins that is part of the OpenFramework project, with ofxAndroid being another. The OpenFramework project focuses on easy implementation of sensor data interpretation for robotics running embedded devices and has many features such as face recognition, body tracking, and gesture detection. While this kind of framework may seem overly complex as our first step, it also gives plenty of room for future expansion in terms of input detection (e.g. hand gestures) beyond our simple depth-based input (see code description later on). In addition, the fact that there were already documentation available for interfacing the Kinect with Android via OpenFramework (see norituna's work) made setup easy and thus our final choice.

c) Game Engine

StepMania:
http://code.google.com/p/sm-ssc/
http://www.stepmania.com/wiki/Downloads
StepMania is one of the most popular, open source DDR (Dance Dance Revolution) clones still in development today. The version that we wanted to run was a fork called sm-ssc, which included the 5-button "pump" mode that mimicked the "Pump It Up" game style that we originally wanted when the project was named "Kinect It Up". While the prebuilt binaries ran smoothly on Windows 7, the menu animations were extremely choppy when run on Linux. At first we thought it was a by-product of the binary not being optimized, so we recompiled it on Philip's Fedora laptop and ran it. The resulting binaries for both sm-ssc and StepMania 3.9 (the last stable official StepMania build) still ran extremely slow.

We suspected that the problem may be due to the excessive feature overload of the newer versions, so we looked into older versions. Turns out, Linux support was only added somewhere between 3.9 and 3.0 as the next oldest version, StepMania 3.0, did not have Linux support. Talking with the official StepMania developers on #stepmania-devs @ freenode.net IRC channel confirmed that StepMania was never optimized to run fast and that Linux support was added but not the main target platform. Regardless, we compiled sm-ssc on the Beagleboard overnight. The resulting game was not just slow; it was completely unplayable at less than 5 FPS in the menu and less than 10 FPS in-game. As much as we liked StepMania, the Linux version was way too unoptimized to be considered a playable.

FlashFlashRevolution:
http://www.flashflashrevolution.com/

FlashFlashRevolution is another DDR clone but made to run inside your web browser. It relies on the web browser being capable of running Adobe Flash and taking keyboard inputs. Flash support was available on the Beagleboard and having the Kinect send keyboard inputs would just require hacking up a special software driver. Because most of the processing is handled on the FFR server, there is little computational strain for the Beagleboard; however, this would require the Beagleboard to have steady internet connectivity. Not only did we find our ethernet connection choppy and unreliable, but our university's wired authentication system (UPenn's NetReg) required us to reregister every time we lost connectivity. In Android, ever time we replugged in the ethernet port into the board, a new MAC address would be assigned to it, so we had to wait out the three minute registration time instead. Due to this, we ended up forgoing the FFR option.

Beats, Advanced Rhythm Game:
http://beatsportable.com/
https://market.android.com/details?id=com.beatsportable.beats

Beats is an Android rhythm game developed by one of the authors, Philip Peng. While the project is closed source, it was written 100% from scratch so Philip was able to easily modify it to add Kinect support. Because of its closed-source nature, it the last considered option. Since StepMania ran terribly slow and internet connectivity was too unreliable for FlashFlashRevolution, we ended up deciding to go ahead with modifying Beats anyway. It turned out to work very well as Philip was also able to directly integrate an actual Kinect FPS counter into the game and built-in tweakable settings.

d) Input Method

Terminal:

The first simple and easy way of achieving our goal was to use the Beagleboard purely as a processing machine and pipe out output through the serial port to a computer. The computer would then read that output from the terminal and send keystrokes/signals to the program running on the computer (StepMania or FFR). This would be easy to do as we had learned how to communicate through the serial port in the past and there would be no major performance issues as the Beagleboard would do nothing but calculations. While easy, we ultimately decided against this approach as we found the process to lack any real “embedded” aspect. Thus, we considered the approach only to be a backup plan if all else failed.

USB Emulation:

USB emulation was our initial idea. Instead of plugging in a Kinect into a computer and requiring necessary drivers/software installed on the host to communicate with it, the user would just plug in the Beagleboard (which has the Kinect attached) and it would act like a standard keyboard. While the idea sounded cool at first, it turned out to be a lot more complicated than we though.

One way to accomplish USB keyboard emulation would be to reverse engineer the USB-On-The-Go controller that the Beagleboard uses and hack that code to replicate the signals sent out by a commercial keyboard. Not only does that require in-depth knowledge of how the Beagleboard’s USB-OTG controller works, but it also requires us to reverse engineer a commercial keyboard that would likely be supported by the host’s operating system. Since neither of those processes are well documented (publicly) and neither of us had any of that experience, we looked at the other alternative: emulation through hardware. It turns out that hardware emulation wasn’t that difficult and only required either taking apart some old pre-existing keyboards or buying a dedicated controller made for the job (e.g. the UltriMarc’s I-PAC). Unfortunately, at the time when we discovered this, Eric (our EE-person) was sick and the board would arrive too late to be our main focus. So, instead, we decided to experiment with just the hardware we had on hand.

Beagleboard:

Our last option (and preferred method) was to run the entire thing on the Beagleboard. Kinect input-processing, game running, and even video output would have to all run smoothly on the Beagleboard without being too out-of-sync. While this would fully fit in with the “embedded” part of the course, we were worried that the board would be too slow. This was our first concern when we compiled and ran StepMania on the Beagleboard and discovered the game to be unplayable even with a standard USB keyboard. Fortunately, Beats ran plenty fine on the Beagleboard, so we stuck with that.

e) Framerate Estimations

Due to differences between the platforms and setups, we were unable to accurately measure the efficiency of each of the platforms/setups we used. The below is an comparison of the various framerates obtained through our various tests. Some of these values were collected while others were approximation: interpret values only relative to each other:

	libfreenect text-only	libfreenect graphical	sm-ssc*	StepMania 3.9*	Beats*	Beats with Kinect**
PC (Fedora)	60 FPS	60 FPS	5-10 FPS	15 FPS	N/A	N/A
Beagleboard (Ubuntu)	25-30 FPS	20-25 FPS	1-5 FPS	N/A	N/A	N/A
Beagleboard (Android)	20 FPS	15-20 FPS	N/A	N/A	50-55 FPS	10-15 FPS

* = without Kinect support

** = Beats itself still ran at 50 FPS

8. Code Explanations

All relevant source code and diffs can be found in the `src/` folder.

a) ofsample Modifications

The modified ofsample works as follows. Depth data is read in from the Kinect. The Kinect has a 640x480 field of vision and returns an array of floats representing distance data. That 640x480 array is divided up into 16x12 blocks of 40 pixel x 40 pixel boxes.

Every one of the 1600 pixels inside each box is checked to see if they are of a depth value beyond a certain threshold (we decided experimentally that 100cm was a good distance). If more than half of the pixels in a box is beyond the threshold, the box is counted as "pushed" and a counter increased. The 16 columns of boxes correspond to the DDR arrow keys. The first four columns are responsible for "left", the next four for "down", next four for "up", last four for "right". When a box in the correct column is "pushed", the corresponding arrow counter is increased.

A modified depth image is drawn to the screen with each individual pixel coloured black if they are closer than 100cm from the Kinect and yellow if they are farther than that. Below the depth image is displayed the numerical count values, the random raw depth value of a random pixel (pixel 200, which is the 201st pixel from the left in the very top row of the Kinect's vision), and the framerate that the Kinect is updating at.

Using this data, we were able to estimate a good numerical value to compare to for determining whether or not a full arrow "press" was executed (e.g. the player pushed their hand forward in the air, pressing an imaginary button). Each count value falls between 0 and 48 (4x12 = 48), and the final minimum value we agreed as being a clear arrow "press" was 20 when standing ~125 cm away from the Kinect.

b) ofsample Code

testApp.cpp

```
#include "testApp.h"


//--------------------------------------------------------------
void testApp::setup()
{
    kinect.init();
    kinect.setUseRGBCamera(false);
    kinect.setVerbose(false);
    kinect.open();

    grayImage.allocate(kinect.width, kinect.height);
    ofSetFrameRate(10);
}

//--------------------------------------------------------------
void testApp::update()
{
    ofBackground(100, 100, 100);
    kinect.update();

    // 640 x 480
    // 4 x (4 x 12 -> 40p x 40p)
    // Each count can max at 48
    // 6 is our experimentally determined threshold of an acceptable "press"
    float *distancePixels = kinect.getDistancePixels();

    count_left = 0;
    for (int column = 0; column < 4; column++) {
     for (int row = 0; row < 12; row++) {
            int count = 0;
            for (int y = 0; y < 40; y++) {
                   for (int x = 0; x < 40; x++) {
                          int i = (row * 40 + y) * 640 + column * 40 + x;
                          if(distancePixels[i] < 100) count++;
                   }
            }
            if (count > 800) { // More than half
                   count_left++;
            }
     }
    }

    count_down = 0;
    for (int column = 0; column < 4; column++) {
     for (int row = 0; row < 12; row++) {
            int count = 0;
            for (int y = 0; y < 40; y++) {
                   for (int x = 0; x < 40; x++) {
                          int i = (row * 40 + y) * 640 + column * 40 + x + 160 * 1;
                          if(distancePixels[i] < 100) count++;
                   }
            }
            if (count > 800) { // More than half
                   count_down++;
            }
     }
    }

    count_up = 0;
    for (int column = 0; column < 4; column++) {
     for (int row = 0; row < 12; row++) {
            int count = 0;
            for (int y = 0; y < 40; y++) {
                   for (int x = 0; x < 40; x++) {
                          int i = (row * 40 + y) * 640 + column * 40 + x + 160 * 2;
                          if(distancePixels[i] < 100) count++;
                   }
            }
            if (count > 800) { // More than half
                   count_up++;
            }
     }
    }

    count_right = 0;
    for (int column = 0; column < 4; column++) {
     for (int row = 0; row < 12; row++) {
            int count = 0;
            for (int y = 0; y < 40; y++) {
                   for (int x = 0; x < 40; x++) {
                          int i = (row * 40 + y) * 640 + column * 40 + x + 160 * 3;
                          if(distancePixels[i] < 100) count++;
                   }
            }
            if (count > 800) { // More than half
                   count_right++;
            }
     }
    }


    // For graphics
    grayImage.setFromPixels(kinect.getDepthPixels(), kinect.width, kinect.height);
    unsigned char * pix = grayImage.getPixels();
    int numPixels = grayImage.getWidth() * grayImage.getHeight();
    for(int i = 0; i < numPixels; i++){
     if(distancePixels[i] < 100){
            pix[i] = 255;
            }else{
                   pix[i] = 0;
            }
           }
      }

      //--------------------------------------------------------------
      void testApp::draw()
      {

          ofSetColor(255, 255, 0); // yellow
          grayImage.draw(0, 0, 640, 480);


          ofSetColor(155, 155, 255);
          char reportStr[1024];
          float *distancePixels = kinect.getDistancePixels();
          sprintf(
                 reportStr,
                 "L %d / D %d / U %d / R %d / dist[200] %f / fps: %f",
                  count_left, count_down, count_up, count_right, distancePixels[200], ofGetFrameRate()
                  );
          ofDrawBitmapString(reportStr, 10, 270);
      }

      //--------------------------------------------------------------
      void testApp::exit(){
          kinect.close();
          kinect.clear();
      }

      //--------------------------------------------------------------
      void testApp::keyPressed (int key)
      {}

      //--------------------------------------------------------------
      void testApp::mouseMoved(int x, int y)
      {}

      //--------------------------------------------------------------
      void testApp::mouseDragged(int x, int y, int button)
      {}

      //--------------------------------------------------------------
      void testApp::mousePressed(int x, int y, int button)
      {}

      //--------------------------------------------------------------
      void testApp::mouseReleased(int x, int y, int button)
      {}

      //--------------------------------------------------------------
      void testApp::windowResized(int w, int h)
      {}

```

testapp.h

```
#ifndef _TEST_APP
#define _TEST_APP

#include "ofMain.h"

#include "ofxOpenCv.h"
#include "ofxKinect.h"


class testApp : public ofBaseApp
{

    public:

    void setup();
    void update();
    void draw();
    void exit();

    //void drawPointCloud();

    void keyPressed  (int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);

    ofxKinect kinect;

    ofxCvGrayscaleImage     grayImage;

    int count_left;
    int count_down;
    int count_up;
    int count_right;
};

#endif

```

c) ofsample Code Explanations

The only two files modified from the ofsample source code was testApp.cpp and testApp.h. The only modification to testApp.h was cleanup of some unused lines and the addition of the int variables count_left, count_down, count_up, and count_right. The following are explanations for the code in testApp.cpp.

The testApp::setup() function is called by main.cpp and executes the initial setup for the Kinect. It is unmodified compared to the original testApp.cpp. Lines 7-10 initialize the Kinect with appropriate parameters (do not read RGB data, only depth data). Line 12 allocates appropriate memory for the graphical pixel array grayImage that will be used for drawing purposes later. Line 13 supposedly sets the frame update rate for the OpenFramework internal update thread but I have not noticed it to actually effect the framework of the application on either PC or Beagleboard, so I left it as its default value (10).

The testApp::update() function is called periodically by the OpenFramework internal update thread. Lines 19 and 20 clear the memory buffers and tells the Kinect to capture new data. Line 26 creates a float pointer that points to the distance data array from the Kinect. Line 28 resets the count_left counter to 0, and lines 29 and 30 specify a loop over all the boxes in the first four columns, which are designated to the left arrow. Line 31 creates a new count variable for keeping track of the number of pixels that have been "pressed", and lines 32 and 33 specify a loop over the 1600 pixels in each box. Line 34 calculates the exact index of the targetted pixel via its box row number, box column number, and x and y coordinates within the box. Line 35 increases the count variable if the targetted pixel's distance value is under 100, e.g. the distance of the object at that point in the Kinect's view is closer than 100cm from the Kinect's distance camera. Lines 38-40 increments the count_left counter if over half (800) of the pixels in the checked box is "pressed". Lines 28-42 are then repeated for count_down, count_up, and count_right with their corresponding box columns (note the constant that is multiplied by 160 in each line 35 counterpart).

Line 94 sets grayImage's pixel data to the Kinect's depth pixels. I am not 100% sure the distinction between the Kinect's "distance" and "depth" data, but I assume that they ultimately represent the same thing, only in different formats. It seems that the distance data are floats representing centimeter distance from the Kinect camera whereas depth deata are grayscale pixel values corresponding to that distance). Lines 95-104 iterate over each pixel and sets the pixel to either 255 (black) if the corresponding distance value is over 100, or 0 (white) if not.

The testApp::draw() function is also called periodically by the OpenFramework internal update thread, most likely right after a testApp::update() call. Line 110 clears the screen's drawing box's background to yellow. Line 111 then draws black ontop of the yellow background whenever the corresponding pixel is 255 (black). Lines 114 to 122 then draw the values of count_left, count_down, count_up, count_up, the distance value of pixel 200, and finally the update framerate (which I assume is the same as the rate of Kinect updating and data processing).

The remaining functions are all dummies and not used at all (but required to be implemented as an OpenFramework app).

d) Beats Modifications

Beats was modified to include Kinect support as an extra input source that is checked against every frame. The ofsample demo code was modified and dummied in parts such that the Beats code could call a function to collect Kinect distance data, update only the arrow counts from that data, then return a value telling Beats whether or not the corresponding arrow is currently being “pressed” or not. Unlike the ofsample app, there is no graphical displaying of the Kinect’s data (for speed reasons), so it is assumed that the player has already calibrated his/her positioning relative to the Kinect based on prior running of the ofsample app.

For speed reasons (and to eliminate false presses above or below the player’s abdomen area), only the middle half of the Kinect’s view is checked for pressed pixels. The distance threshold is still hardcoded to 100cm, but a count threshold is introduced for comparison for determining whether or not the entire arrow has been pressed. The threshold value is adjustable by a setting within Beats, and defaults to 15 (but Eric found 25 to be a better value during our demos). Because it is expected that the side arrows be harder to reach, the count thresholds for them are ¾ of the normal value.

Because Beats updates at a faster rate (50+ FPS) than the Kinect data can be processed (10-15 FPS), the request for input from the Kinect is sent in a separate, mutex’ed thread. While this slows down the framerate by -1-2 FPS, it allows Beats to run very smoothly. In addition, the actual framerate values are calculated and displayed graphically.

e) Beats Code

beats-r475-kinect.diff:

```
Index: beats/default.properties
===================================================================
--- beats/default.properties    (revision 475)
+++ beats/default.properties    (working copy)
@@ -8,4 +8,4 @@
 # project structure.

 # Project target.
-target=android-10
+target=android-8
Index: beats/jni/testApp.cpp
===================================================================
--- beats/jni/testApp.cpp    (revision 0)
+++ beats/jni/testApp.cpp    (revision 0)
@@ -0,0 +1,142 @@
+#include "testApp.h"
+
+void testApp::setup2()
+{
+    kinect.init();
+    kinect.setUseRGBCamera(false);
+    kinect.setVerbose(false);
+    kinect.open();
+}
+
+void testApp::setThresh(int thresh)
+{
+    kinectThresh = thresh;
+}
+
+void testApp::update2()
+{
+    kinect.update();
+    distancePixels = kinect.getDistancePixels();
+    count_tap = 0;
+    for (int column = 0; column < 4; column++) {
+           for (int row = 3; row < 9; row++) {
+                  int count = 0;
+                  for (int y = 0; y < 40; y++) {
+                         for (int x = 0; x < 40; x++) {
+                                int i = (row * 40 + y) * 640 + column * 40 + x;
+                                if(distancePixels[i] < 100) count++;
+                         }
+                  }
+                  if (count > 800) { // More than half
+                         count_tap++;
+                  }
+           }
+    }
+    if (count_tap > (kinectThresh * 3) / 4) {
+           left = 1; // left
+    } else {
+           left = 0;
+    }
+    count_tap = 0;
+    for (int column = 0; column < 4; column++) {
+           for (int row = 3; row < 9; row++) {
```

```
+            int count = 0;
+            for (int y = 0; y < 40; y++) {
+                for (int x = 0; x < 40; x++) {
+                    int i = (row * 40 + y) * 640 + column * 40 + x + 160 * 3;
+                    if(distancePixels[i] < 100) count++;
+                }
+            }
+            if (count > 800) { // More than half
+                count_tap++;
+            }
+        }
+    }
+    if (count_tap > (kinectThresh * 3) / 4) {
+        right = 1;
+    } else {
+        right = 0;
+    }
+    count_tap = 0;
+    for (int column = 0; column < 4; column++) {
+        for (int row = 3; row < 9; row++) {
+            int count = 0;
+            for (int y = 0; y < 40; y++) {
+                for (int x = 0; x < 40; x++) {
+                    int i = (row * 40 + y) * 640 + column * 40 + x + 160 * 1;
+                    if(distancePixels[i] < 100) count++;
+                }
+            }
+            if (count > 800) { // More than half
+                count_tap++;
+            }
+        }
+    }
+    if (count_tap > kinectThresh) {
+        down = 1;
+    } else {
+        down = 0;
+    }
+    count_tap = 0;
+    for (int column = 0; column < 4; column++) {
+        for (int row = 3; row < 9; row++) {
+            int count = 0;
+            for (int y = 0; y < 40; y++) {
+                for (int x = 0; x < 40; x++) {
+                    int i = (row * 40 + y) * 640 + column * 40 + x + 160 * 2;
+                    if(distancePixels[i] < 100) count++;
+                }
+            }
+            if (count > 800) { // More than half
+                count_tap++;
+            }
+        }
+    }
+    if (count_tap > kinectThresh) {
+        up = 1;
+    } else {
+        up = 0;
+    }
+}
+
+//--------------------------------------------------------------
+void testApp::setup()
+{}
+
+//--------------------------------------------------------------
+void testApp::update()
+{}
+
+//--------------------------------------------------------------
+void testApp::draw()
+{}
+
+//--------------------------------------------------------------
+void testApp::exit(){
+    kinect.close();
+    kinect.clear();
+}
+
+//--------------------------------------------------------------
+void testApp::keyPressed (int key)
+{}
+
+//--------------------------------------------------------------
+void testApp::mouseMoved(int x, int y)
+{}
+
+//--------------------------------------------------------------
+void testApp::mouseDragged(int x, int y, int button)
+{}
+
+//--------------------------------------------------------------
+void testApp::mousePressed(int x, int y, int button)
+{}
+
+//--------------------------------------------------------------
+void testApp::mouseReleased(int x, int y, int button)
+{}
+
+//--------------------------------------------------------------
+void testApp::windowResized(int w, int h)
+{}
Index: beats/jni/main.cpp
===================================================================
--- beats/jni/main.cpp    (revision 0)
+++ beats/jni/main.cpp    (revision 0)
@@ -0,0 +1,56 @@
+#include "ofMain.h"
+#include "testApp.h"
+#include <jni.h>
+#include "ofAppAndroidWindow.h"
+
+//========================================================================
+extern "C"{
+testApp *kinectApp;
+
+// UNUSED
+void Java_cc_openframeworks_OFAndroid_init( JNIEnv*  env, jobject  thiz ){
+}
+
+void Java_cc_openframeworks_OFAndroid_setup2( JNIEnv*  env, jobject  thiz ){
+    kinectApp = new testApp();
+    kinectApp->setup2();
+}
+
+void Java_cc_openframeworks_OFAndroid_setThresh( JNIEnv*  env, jobject  thiz, jint thresh){
+    kinectApp->setThresh(thresh);
+}
+
+void Java_cc_openframeworks_OFAndroid_update2( JNIEnv*  env, jobject  thiz, jint thresh){
+    kinectApp->update2();
+}
+
+JNIEXPORT
+jint
+JNICALL Java_cc_openframeworks_OFAndroid_left( JNIEnv*  env, jobject  thiz, jint thresh){
+    return kinectApp->left;
+}
+
+JNIEXPORT
+jint
+JNICALL Java_cc_openframeworks_OFAndroid_down( JNIEnv*  env, jobject  thiz, jint thresh){
+    return kinectApp->down;
+}
+
+JNIEXPORT
+jint
+JNICALL Java_cc_openframeworks_OFAndroid_up( JNIEnv*  env, jobject  thiz, jint thresh){
+    return kinectApp->up;
+}
+
+JNIEXPORT
+jint
+JNICALL Java_cc_openframeworks_OFAndroid_right( JNIEnv*  env, jobject  thiz, jint thresh){
+    return kinectApp->right;
+}
+
+}
+
+// UNUSED
+int main(){
+    return 0;
+}
Index: beats/jni/Android.mk
===================================================================
--- beats/jni/Android.mk    (revision 0)
+++ beats/jni/Android.mk    (revision 0)
@@ -0,0 +1,54 @@
+LOCAL_PATH := $(call my-dir)
+
+include $(CLEAR_VARS)
+
+LOCAL_MODULE      := OFAndroidApp
+LOCAL_SRC_FILES := testApp.cpp main.cpp
+
+LOCAL_C_INCLUDES := $(LOCAL_PATH)/../../../build/prebuilt/linux-x86/arm-eabi-4.4.0/arm-eabi/include/c++/4.4.0 \
+                   $(LOCAL_PATH)/../../../build/platforms/android-8/arch-arm/usr/include \
+                   $(LOCAL_PATH)/../../../build/prebuilt/linux-x86/arm-eabi-4.4.0/arm-eabi/include/c++/4.4.0/arm-eabi \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks/app/ \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks/communication \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks/events \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks/graphics \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks/sound \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks/utils \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/freetype/include \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/freetype/include/freetype2 \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/glu/include_android \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/FreeImage/include \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxThread/src \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxVectorMath/src \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxOpenCv/src \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/poco/include \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks/video \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxOpenCv/src/ \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxOpenCv/libs/opencv/include_android/ \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxKinect/libs/includes/ \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxKinect/src/ \
+                   $(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxAndroid/src
+
+LOCAL_CXXFLAGS += -DANDROID -D__ANDROID__ -isystem $(SYSROOT)/usr/include
+LOCAL_CFLAGS += -DANDROID -D__ANDROID__  $(LOCAL_C_INCLUDES:%=-I%)
+
+LOCAL_LDLIBS := -L$(LOCAL_PATH)/../../../build/prebuilt/linux-x86/arm-eabi-4.4.0/arm-eabi/lib/ \
+           -L$(LOCAL_PATH)/../../../build/platforms/android-8/arch-arm/usr/lib \
+           -L$(LOCAL_PATH)/../../../build/prebuilt/linux-x86/arm-eabi-4.4.0/lib/gcc/arm-eabi/4.4.0/ \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/openFrameworks/Debug/ \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/Debug/ \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/poco/lib/android/ \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/FreeImage/lib/android/ \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/
```

```
freetype/lib/android/ \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/libs/glu/lib/android/ \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxOpenCv/libs/opencv/lib/android \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxOsc/libs/oscpack/lib/android/ \
+           -L$(LOCAL_PATH)/../../../../of_preRelease_v0062_android_FAT/addons/ofxKinect/libs/ \
+           -ldl -lstdc++ -lgcc -llog -lm -lc -laddons -loscpack -lopenFrameworks -lPocoUtil \
+           -lPocoFoundation -lfreetype -lfreeimage -lcv -lcxcore -lopencv_lapack -lcvaux -lGLU -lz -lGLESv1_CM
+
+
+
+include $(BUILD_SHARED_LIBRARY)
+
Index: beats/jni/testApp.h
===================================================================
--- beats/jni/testApp.h    (revision 0)
+++ beats/jni/testApp.h    (revision 0)
@@ -0,0 +1,45 @@
+#ifndef _TEST_APP
+#define _TEST_APP
+
+#include "ofMain.h"
+
+#include "ofxOpenCv.h"
+#include "ofxKinect.h"
+
+
+class testApp : public ofBaseApp
+{
+
+    public:
+
+           // New stuff
+           void setup2();
+           void update2();
+           void setThresh(int thresh);
+           int kinectThresh;
+           int count_tap;
+           int left;
+           int down;
+           int up;
+           int right;
+           float *distancePixels;
+
+           // Old dummied stuff
+           void setup();
+           void update();
+           void draw();
+           void exit();
+
+           //void drawPointCloud();
+
+           void keyPressed  (int key);
+           void mouseMoved(int x, int y );
+           void mouseDragged(int x, int y, int button);
+           void mousePressed(int x, int y, int button);
+           void mouseReleased(int x, int y, int button);
+           void windowResized(int w, int h);
+
+           ofxKinect kinect;
+};
+
+#endif
Index: beats/.project
===================================================================
--- beats/.project    (revision 473)
+++ beats/.project    (working copy)
@@ -1,33 +1,40 @@
-<?xml version="1.0" encoding="UTF-8"?>
-<projectDescription>
-    <name>Beats</name>
-    <comment></comment>
-    <projects>
-    </projects>
-    <buildSpec>
-           <buildCommand>
-                   <name>com.android.ide.eclipse.adt.ResourceManagerBuilder</name>
-                   <arguments>
-                   </arguments>
-           </buildCommand>
-           <buildCommand>
-                   <name>com.android.ide.eclipse.adt.PreCompilerBuilder</name>
-                   <arguments>
-                   </arguments>
-           </buildCommand>
-           <buildCommand>
-                   <name>org.eclipse.jdt.core.javabuilder</name>
-                   <arguments>
-                   </arguments>
-           </buildCommand>
-           <buildCommand>
-                   <name>com.android.ide.eclipse.adt.ApkBuilder</name>
-                   <arguments>
-                   </arguments>
-           </buildCommand>
-    </buildSpec>
-    <natures>
-           <nature>com.android.ide.eclipse.adt.AndroidNature</nature>
-           <nature>org.eclipse.jdt.core.javanature</nature>
-    </natures>
-</projectDescription>
+<?xml version="1.0" encoding="UTF-8"?>
+<projectDescription>
+    <name>Beats</name>
+    <comment></comment>
+    <projects>
+    </projects>
+    <buildSpec>
+           <buildCommand>
+                   <name>com.android.ide.eclipse.adt.ResourceManagerBuilder</name>
+                   <arguments>
+                   </arguments>
+           </buildCommand>
+           <buildCommand>
+                   <name>com.android.ide.eclipse.adt.PreCompilerBuilder</name>
+                   <arguments>
+                   </arguments>
+           </buildCommand>
+           <buildCommand>
+                   <name>org.eclipse.jdt.core.javabuilder</name>
+                   <arguments>
+                   </arguments>
+           </buildCommand>
+           <buildCommand>
+                   <name>com.android.ide.eclipse.adt.ApkBuilder</name>
+                   <arguments>
+                   </arguments>
+           </buildCommand>
+    </buildSpec>
+    <natures>
+           <nature>com.android.ide.eclipse.adt.AndroidNature</nature>
+           <nature>org.eclipse.jdt.core.javanature</nature>
+    </natures>
+    <linkedResources>
+           <link>
+                   <name>ofAndroLib_src</name>
+                   <type>2</type>
+                   <location>/home/noritsuna/kinect/of_preRelease_v0062_android_FAT/addons/ofxAndroid/ofAndroidLib/src</location>
+           </link>
+    </linkedResources>
+</projectDescription>
Index: beats/src/com/beatsportable/beats/GUIHandlerTap.java
===================================================================
--- beats/src/com/beatsportable/beats/GUIHandlerTap.java    (revision 473)
+++ beats/src/com/beatsportable/beats/GUIHandlerTap.java    (working copy)
@@ -5,6 +5,9 @@
 import android.graphics.Canvas;
 import android.graphics.Paint;
 import android.graphics.Rect;
+import android.os.AsyncTask;
+import cc.openframeworks.OFAndroid;
+
 import com.beatsportable.beats.DataNote.NoteType;
 import com.beatsportable.beats.GUIScore.*;

@@ -77,6 +80,16 @@
            debugTapboxRects = new Rect[Tools.PITCHES];

            //setupXY(); // Call in GUIGame
+
+           // Kinect
+           System.loadLibrary("OFAndroidApp");
+           OFAndroid.setup2();
+           KinectInput.h = this;
+           KinectInput.kinectThresh = Integer.valueOf(
+                          Tools.getSetting(R.string.kinectDist, R.string.kinectDistDefault));
+           OFAndroid.setThresh(KinectInput.kinectThresh);
+           KinectInput.kinectFrames = 0;
+           KinectInput.kinectFramesTotal = 0;
     }

     @Override
@@ -210,6 +223,9 @@
                    offScreenTime = yToTime(Tools.screen_h);
            }

+           // Kinect
+           new KinectInput().execute();
+
      // TODO
      // I think logic flow of events should be changed so onTouchDown_One
      // combo updating code is here instead of in reaction to the touch event
@@ -223,7 +239,7 @@

              o.onHold(currentTime, score);

-             if (currentTime > o.end_time) {
+             if (currentTime > o.end_time || o.start_time == o.end_time) {
                  AccuracyTypes acc = o.onLastFrame(currentTime, score, true);

                  if (acc != AccuracyTypes.X_IGNORE_ABOVE) {
@@ -523,3 +539,44 @@
     }

  }
+
+class KinectInput extends AsyncTask<Void, Void, Void> {
+    protected static GUIHandlerTap h;
+    protected static int kinectThresh;
+    private static int mutex = 0;
+
+    @Override
+    protected Void doInBackground(Void... arg0) {
+            if (mutex != 1) {
+                    mutex = 1;
+                    OFAndroid.update3();
+                    if (OFAndroid.left() == 1) {
+                            h.onTouch_Down_One(3);
+                    } else {
+                            h.onTouch_Up_One(3);
+                    }
+                    if (OFAndroid.down() == 1) {
+                            h.onTouch_Down_One(2);
+                    } else {
+                            h.onTouch_Up_One(2);
+                    }
+                    if (OFAndroid.up() == 1) {
+                            h.onTouch_Down_One(1);
+                    } else {
+                            h.onTouch_Up_One(1);
+                    }
+                    if (OFAndroid.right() == 1) {
+                            h.onTouch_Down_One(0);
+                    } else {
+                            h.onTouch_Up_One(0);
+                    }
+                    kinectFrames++;
+                    mutex = 0;
+            }
+    }
+
+    public static int kinectFrames = 0;
+    public static int kinectFramesTotal = 0;
+
+}
+
Index: beats/src/com/beatsportable/beats/Tools.java
===================================================================
--- beats/src/com/beatsportable/beats/Tools.java    (revision 473)
+++ beats/src/com/beatsportable/beats/Tools.java    (working copy)
@@ -197,6 +197,7 @@
     }
 
     public static void toast(String msg) {
+            /*
             if (c == null) return;
             try {
                     Toast.makeText(c, msg, Toast.LENGTH_SHORT).show();
@@ -205,9 +206,11 @@
                     ToolsTracker.error("Tools.toast", e, c.getLocalClassName());
             }
             debugLogCat(msg);
+            */
     }
 
     public static void toast_long(String msg) {
+            /*
             if (c == null) return;
             try {
                     Toast.makeText(c, msg, Toast.LENGTH_LONG).show();
@@ -216,6 +219,7 @@
                     ToolsTracker.error("Tools.toast_long", e, c.getLocalClassName()
);
             }
             debugLogCat(msg);
+            */
     }
 
     public static OnClickListener cancel_action = new OnClickListener() {
@@ -239,6 +243,7 @@
                     final int ignoreSetting, boolean checked,
                     boolean cancelable
             ) {
+            /*
             if (c == null) return;
 
             AlertDialog.Builder alertBuilder = new AlertDialog.Builder(c);
@@ -282,6 +287,7 @@
             alertBuilder.show().setOwnerActivity(c);
 
             debugLogCat(msg.toString());
+            */
     }
 
     public static void note(
Index: beats/src/com/beatsportable/beats/MenuHome.java
===================================================================
--- beats/src/com/beatsportable/beats/MenuHome.java    (revision 476)
+++ beats/src/com/beatsportable/beats/MenuHome.java    (working copy)
@@ -33,6 +33,7 @@
     private static Locale defaultLocale;
 
     // AdMob
+    /*
     private AdView adMobView;
     private static final HashSet<String> adMobKeyWordsSet = new HashSet<String>();
     static {
@@ -42,12 +43,14 @@
             adMobKeyWordsSet.add("arcade game");
             adMobKeyWordsSet.add("action game");
```

```
         }
+        */

         // MobFox
-        private MobFoxView mobFoxView;
+        //private MobFoxView mobFoxView;

         // Startup Warnings
         private void versionCheck() {
+                /*
                 if (Integer.parseInt(Build.VERSION.SDK) < 7 &&
                         !Tools.getBooleanSetting(R.string.ignoreLegacyWarning, R.string.ignoreLegacyWarningDefault)) {
                         // Multitouch warning
@@ -56,13 +59,15 @@
                                         Tools.cancel_action, R.string.ignoreLegacyWarning
                                         );
                 }
+                */
         }

         private void updateCheck() {
-                new ToolsUpdateTask().execute(Tools.getString(R.string.Url_version));
+                //new ToolsUpdateTask().execute(Tools.getString(R.string.Url_version));
         }

         private void showBackgroundData() {
+                /*
                 // Background data warning
                 DialogInterface.OnClickListener sync_action = new DialogInterface.OnClickListener() {
                         public void onClick(DialogInterface dialog, int id) {
@@ -90,10 +95,12 @@
                                 cancel_action,
                                 R.string.ignoreSyncWarning
                                 );
+                */
         }

         // On Finish
         private void backgroundDataUncheck() {
+                /*
                 ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
                 if (Tools.disabledBackgroundData && !cm.getBackgroundDataSetting()) {
                         // Background data warning
@@ -130,10 +137,12 @@
                 } else {
                         finish();
                 }
+                */
+                finish();
         }

         private void showNotes() {
-
+                /*
                 // New User notes
                 if (!Tools.getBooleanSetting(R.string.ignoreNewUserNotes, R.string.ignoreNewUserNotesDefault)) {
```

```

@@ -206,6 +215,7 @@
                    // Always make folders
                    if (Tools.isMediaMounted()) Tools.makeBeatsDir();
            }
+           */
     }

     // Activity Result
@@ -256,8 +266,9 @@

     // Update layout images
     private void updateLayout() {
-           updateLanguage();
+           //updateLanguage();

+           /*
            // AdMob
            AdRequest adMobAd = new AdRequest();
            adMobAd.setKeywords(adMobKeyWordsSet);
@@ -266,6 +277,7 @@
            // MobFox
            mobFoxView.pause();
            mobFoxView.resume();
+           */

            // Titlebar
            title = Tools.getString(R.string.MenuHome_titlebar) + " [" + Tools.getString(R.string.App_version) + "]";
@@ -376,13 +388,15 @@
            if (Tools.getBooleanSetting(R.string.resetSettings, R.string.resetSettingsDefault)) {
                    Tools.resetSettings();
            }
+           /*
            if (Tools.getBooleanSetting(R.string.additionalVibrations, R.string.additionalVibrationsDefault)) {
                    ((Vibrator) getSystemService(Context.VIBRATOR_SERVICE)).vibrate(300); // ready to rumble!
            }
+           */
            updateCheck();
            setupLayout();
-           versionCheck();
-           showNotes();
+           //versionCheck();
+           //showNotes();
     }

     private void formatMenuItem(final TextView tv, int text) {
@@ -415,8 +429,9 @@

     private void setupLayout() {
            setContentView(R.layout.main);
-           setVolumeControlStream(AudioManager.STREAM_MUSIC); // To control media volume at all times
+           //setVolumeControlStream(AudioManager.STREAM_MUSIC); // To control media volume at all times

+           /*
            // For ads
            LayoutParams params = new LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT);
         params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
@@ -431,6 +446,7 @@
         adMobView = new AdView(this, AdSize.BANNER, getResources().getString(R.string.AdMob_key));
         adMobView.setGravity(RelativeLayout.ALIGN_PARENT_BOTTOM);
         layout.addView(adMobView, params);
+        */
 
         backgroundRes = -1; // Force background reload
         updateLayout();
@@ -481,11 +497,11 @@
         TextView select_song_b = (TextView) findViewById(R.id.select_song);
         select_song_b.setOnClickListener(new OnClickListener() {
             public void onClick(View v) {
-                if (Tools.isMediaMounted()) {
+                //if (Tools.isMediaMounted()) {
                     Intent i = new Intent();
                     i.setClass(MenuHome.this, MenuFileChooser.class);
                     startActivityForResult(i, SELECT_MUSIC);
-                }
+                //}
             }
         });
 
@@ -495,6 +511,7 @@
             public void onClick(View v) {
                 Intent i = new Intent();
                 i.setClass(MenuHome.this, MenuSettings.class);
+                //i.setClass(MenuHome.this, OFsample2.class);
                 startActivity(i);
             }
         });
Index: beats/src/com/beatsportable/beats/GUIGame.java
===================================================================
--- beats/src/com/beatsportable/beats/GUIGame.java    (revision 473)
+++ beats/src/com/beatsportable/beats/GUIGame.java    (working copy)
@@ -245,7 +245,9 @@
         private long fpsStartTime = 0;
         private long fpsTotalTime = 0;
         private double fps = 0;
+        private double fpsK = 0;
         private double fpsTotal = 0;
+        private double fpsTotalK = 0;
         private String fpsTruncated = "";
         private String fpsTruncatedTotal = "";
         private boolean fpsTotalStarted = false;
@@ -625,19 +627,22 @@
                 if (frameCount > 10) {
                     long fpsCurrentTime = SystemClock.elapsedRealtime();
                     fps = ((double)(frameCount) / ((double)(fpsCurrentTime - fpsStartTime) / 1000d));
+                    fpsK = ((double)(KinectInput.kinectFrames) / ((double)(fpsCurrentTime - fpsStartTime) / 1000d));
                     // Truncate to 2 decimal places
-                    fpsTruncated = String.format("%.2f", fps);
+                    fpsTruncated = String.format("%.2f %.2f", fps, fpsK);
                     if (fpsTotalStarted) {
                         frameCountTotal += frameCount;
```

```
+                                               KinectInput.kinectFramesTotal += KinectInput.kinectFrames;
                                                fpsTotalTime += fpsCurrentTime - fpsStartTime;
                                                fpsTotal = ((double)(frameCountTotal) / ((double)(fpsTotalTime) / 1000d));
-                                               fpsTruncatedTotal = String.format("%.2f", fpsTotal);
+                                               fpsTotalK = ((double)(KinectInput.kinectFramesTotal) / ((double)(fpsTotalTime) / 1000d));
+                                               fpsTruncatedTotal = String.format("%.2f %.2f", fpsTotal, fpsTotalK);
                                        } else {
                                                if (fps > 0 && fps < 99) { // Sanity
                                                        fpsTotalStarted = true;
                                                }
                                        }
-
+                                       KinectInput.kinectFrames = 0;
                                        frameCount = 0;
                                        fpsStartTime = SystemClock.elapsedRealtime();
                                }
Index: beats/res/raw/samples.zip
===================================================================
Cannot display: file marked as a binary type.
svn:mime-type = application/octet-stream
Index: beats/res/values/arrays_const.xml
===================================================================
--- beats/res/values/arrays_const.xml    (revision 473)
+++ beats/res/values/arrays_const.xml    (working copy)
@@ -8,6 +8,17 @@
                <item>1</item>
        </string-array>

+    <string-array name="kinectDistValues">
+               <item>5</item>
+               <item>10</item>
+               <item>15</item>
+               <item>20</item>
+               <item>25</item>
+               <item>30</item>
+               <item>35</item>
+               <item>40</item>
+    </string-array>
+
        <string-array name="difficultyLevelValues">
                <item>0</item>
                <item>1</item>
Index: beats/res/values/settings_trans.xml
===================================================================
--- beats/res/values/settings_trans.xml    (revision 476)
+++ beats/res/values/settings_trans.xml    (working copy)
@@ -71,6 +71,8 @@
      <string name="backgroundImageDesc">"Background image for main menu and stepfiles without one"</string>
      <string name="orientationTitle">"Screen Orientation"</string>
      <string name="orientationDesc">"Default screen orientation during gameplay"</string>
+  <string name="kinectDistTitle">"Kinect Distance"</string>
+  <string name="kinectDistDesc">"Tap distance threshold"</string>
      <string name="tapboxLayoutTitle">"Tapbox Layout"</string>
      <string name="tapboxLayoutDesc">"Arrangement of tapboxes"</string>
```

```
   <string name="tapboxFadingTitle">"Tapbox Fading"</string>
Index: beats/res/values/constants.xml
===================================================================
--- beats/res/values/constants.xml    (revision 476)
+++ beats/res/values/constants.xml    (working copy)
@@ -78,6 +78,8 @@
     <string name="backgroundImageDefault">"blue"</string>
     <string name="orientation">"orientation"</string>
     <string name="orientationDefault">"2"</string>
+    <string name="kinectDist">"kinectDist"</string>
+    <string name="kinectDistDefault">"15"</string>
     <string name="tapboxLayout">"tapboxLayout"</string>
     <string name="tapboxLayoutDefault">"standard"</string>
     <string name="tapboxFading">"tapboxFading"</string>
Index: beats/res/xml/settings.xml
===================================================================
--- beats/res/xml/settings.xml    (revision 476)
+++ beats/res/xml/settings.xml    (working copy)
@@ -130,6 +130,13 @@
     </PreferenceCategory>
     <PreferenceCategory android:title="@string/tapSettingsCategoryTitle">
             <ListPreference
+                   android:title="@string/kinectDistTitle"
+                   android:summary="@string/kinectDistDesc"
+                   android:key="@string/kinectDist"
+                   android:defaultValue="@string/kinectDistDefault"
+                   android:entries="@array/kinectDistValues"
+                   android:entryValues="@array/kinectDistValues" />
+           <ListPreference
                    android:title="@string/tapboxFadingTitle"
                    android:summary="@string/tapboxFadingDesc"
                    android:key="@string/tapboxFading"
Index: beats/.classpath
===================================================================
--- beats/.classpath    (revision 475)
+++ beats/.classpath    (working copy)
@@ -1,10 +1,11 @@
-<?xml version="1.0" encoding="UTF-8"?>
-<classpath>
-    <classpathentry kind="con" path="com.android.ide.eclipse.adt.ANDROID_FRAMEWORK"/>
-    <classpathentry kind="src" path="src"/>
-    <classpathentry kind="src" path="gen"/>
-    <classpathentry kind="lib" path="libs/GoogleAdMobAdsSdk-4.0.4.jar"/>
-    <classpathentry kind="lib" path="libs/mobfoxsdk-1.1.jar"/>
-    <classpathentry kind="lib" path="libs/UHL.jar"/>
-    <classpathentry kind="output" path="bin"/>
-</classpath>
+<?xml version="1.0" encoding="UTF-8"?>
+<classpath>
+    <classpathentry kind="con" path="com.android.ide.eclipse.adt.ANDROID_FRAMEWORK"/>
+    <classpathentry kind="src" path="src"/>
+    <classpathentry kind="src" path="ofAndroLib_src"/>
+    <classpathentry kind="src" path="gen"/>
+    <classpathentry kind="lib" path="libs/GoogleAdMobAdsSdk-4.0.4.jar"/>
+    <classpathentry kind="lib" path="libs/mobfoxsdk-1.1.jar"/>
+    <classpathentry kind="lib" path="libs/UHL.jar"/>
+    <classpathentry kind="output" path="bin"/>
+</classpath>
Index: beats/AndroidManifest.xml
===================================================================
--- beats/AndroidManifest.xml    (revision 476)
+++ beats/AndroidManifest.xml    (working copy)
@@ -3,12 +3,10 @@
     xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.beatsportable.beats"
     android:versionName="@string/App_version"
-    android:versionCode="13"
-    android:installLocation="preferExternal"
-    >
+    android:versionCode="13" android:installLocation="auto">
     <uses-sdk
             android:minSdkVersion="3"
-            android:targetSdkVersion="10"
+            android:targetSdkVersion="8"
     />
     <!-- TODO:
              android:anyDensity is set to false to have scaled, uniform appearance
@@ -19,18 +17,15 @@
             android:smallScreens="true"
             android:normalScreens="true"
             android:largeScreens="true"
-            android:xlargeScreens="true"
             android:anyDensity="false"
             android:resizeable="true"
     />
-    <uses-permission android:name="android.permission.INTERNET"/> <!-- Web-browser, AdMob, MobFox, Localytics -->
-    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/> <!-- AdMob, MobFox -->
-    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/> <!-- MobFox -->
-    <uses-permission android:name="android.permission.READ_PHONE_STATE"/> <!-- MobFox -->
     <uses-permission android:name="android.permission.VIBRATE"/> <!-- Game -->
     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/> <!-- Updates -->
+    <uses-permission android:name="android.permission.CAMERA"/>
     <!-- This is probably deprecated after 1.6 but try it anyway -->
     <uses-permission android:name="android.permission.RAISED_THREAD_PRIORITY"/> <!-- Game -->
+
     <application
             android:enabled="true"
             android:icon="@drawable/icon"
Index: beats/Application.mk
===================================================================
--- beats/Application.mk    (revision 0)
+++ beats/Application.mk    (revision 0)
@@ -0,0 +1,3 @@
+APP_PROJECT_PATH := $(call my-dir)
+LOCAL_MODULE      := OFAndroidApp
+LOCAL_SRC_FILES := testApp.cpp main.cpp
\ No newline at end of file


```

f) Beats Code Explanation

Much of the relevant code works the same way as ofsample and those not relevant are specific to Android development or Beats itself, so these descriptions are brief.

In beats/jni/testApp.cpp. testApp::setup2() initializes just the Kinect with no graphics or any OpenFramework component. testApp::setThresh(int thresh) sets the kinectThreshold value based off the setting in Beats. testApp::update2() does the same thing as in the modified ofsample but also sets the left, down, up, and right values to either 0 for unpressed or 1 for pressed. Instead of count_ counters, a single count_tap counter is recycled throughout. As well, rather than checking all the row boxes, only the middle half is checked (e.g. rows 3-9)The rest of the functions are dummied.

In beats/jni/main.cpp, new Java_cc_openframeworks_OFAndroid_* JNI functions setup2, setThresh, update2, left, down, up and right are added. The corresponding function prototypes and variable declarations are also added to beats/jni/testApp.h and the corresponding public static native functions are also added to the linked ofAndroidLib library's source code.

The file beats/jni/testApp.h has been modified to include the setup2, setThresh and update2 function prototypes and the count_tap, left, down, up and right int counters.

In beats/src/com/beatsportable/beats/GUIHandlerTap.java, the OFAndroidApp (ofAndroidLib) library is loaded, the Kinect initialized, and the threshold value set, then the frame counters reset. The new KinectInput().execute(); line, which is called during the Beat’s normal frame update function, starts a mutex'd AsyncTask (threaded task) be executed. The KinectInput task first obtains the mutex variable (by setting it to 1), tells the Kinect to update, calls the left, down, up, and right functions to check whether each of those values are set or not, then calls the onTouch_Down_One or onTouch_Up_One functions to either press down or lift up the corresponding arrow keys. Once this is done, the kinectFrame count is incremented and the mutex released by being set back to 0.

The files beats/src/com/beatsportable/beats/Tools.java and beats/src/com/beatsportable/beats/MenuHome.java have been modified to remove popup messages and ads for speed.

The file beats/src/com/beatsportable/beats/GUIGame.java has been modified to include FPS values for the Kinect input. New variables fpsK and fpsTotalK are used to calculate the number of frames of Kinect data were collected per second and total average Kinect

FPS. The method of calculation is the same as the normal FPS counters and printed out beside the normal FPS values.

The files beats/res/values/arrays_const.xml, beats/res/values/settings_trans.xml, beats/res/values/constants.xml and beats/res/xml/settings.xml were modified to include a “Kinect Distance” setting in the normal settings menu with possible values between 5 and 40 in increments of 5. This setting is used earlier to set the kinectThreshold value.

The remaining file modifications allow the JNI code to be compiled in and accommodate the new file additions or changes.

9. Future Changes

The main idea behind this final project is to eliminate the physical buttons in music rhythm games and allow the players to wave their hands in the air for gameplay.
The successful implement of our project promise great future potential for further development.

We've managed to accomplish user inputs from four imaginary boxes in front of a player. Knowing the capability of Kinect to detect specific hand gestures or movements, we think it's very possible to implement a music rhythm game that recognizes actual dance moves with hands.

This could eventually even be integrated with dance dance revolution's dance pads to create a much more dynamic gameplay, utilizing a player's both arms and legs.

10. Thanks and Credits

Dr. Rahul Mangharam for being a great professor.

Kevin Conley and Varun Sampath for being fantastic and helpful TAs with almost all of our problems.

Everyone else in Spring 2011 ESE 350 for cool classmates with awesome projects!

Complete list of links/resources (in random order):

- http://www.noritsuna.com/archives/2011/01/openframeworks_kinect_android.html
- http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/TI_Android_DevKit/02_00_00/index_FDS.html
- http://tirokartblog.wordpress.com/2011/01/21/kinect-working-on-ubuntu-10-10/
- http://tirokartblog.wordpress.com/2011/01/22/kinect-on-the-beagleboard-xm-not-working-yet/
- https://wiki.ubuntu.com/ARM/OMAPMaverickInstall
- http://www.flashflashrevolution.com/
- http://code.google.com/p/sm-ssc/
- http://www.stepmania.com/wikiDownloads
- http://www.openframeworks.cc/
- http://www.openni.org/
- http://openkinect.org/wiki/Getting_Started
- https://github.com/OpenKinect/libfreenect
- http://code.google.com/p/rowboat/
- http://elinux.org/BeagleBoardUbuntu
- http://beagleboard.org/project/angstrom/
- http://developer.android.com/guide/publishing/app-signing.html

For more info, check out:

- http://www.seas.upenn.edu/~ese350/
- http://dancewithyourhands.blogspot.com/
- http://beatsportable.com/
- http://beatsportable.com/2011/05/beats-kinect-win/
- http://beatsportable.com/forum/viewtopic.php?f=4&t=1023
- https://market.android.com/details?id=com.beatsportable.beats
- http://www.youtube.com/watch?v=o1MsL-VbyPg

All files (including this one) can be found at:

- http://beatsportable.com/static/kinect/