

# LOGS

## The Portage Handbook

*We acknowledge the financial support of the Government of Canada through the Department of Canadian Heritage (name of program if space permits).*

*Nous reconnaissons l'appui financier du gouvernement du Canada par l'entremise du ministère du Patrimoine canadien (nom du programme, si l'espace le permet).*

**Canada**

**Principal Investigators:**

Prof. Geoffrey Shea (OCAD)

Dr. Paula Gardner (OCAD)

**Researchers:**

Patricio Davila (OCAD/Ryerson)

Michel Blondeau (eentricarts, inc.)

Dr. David McIntosh (OCAD)

Ken Leung (OCAD)

Rob King (Ryerson)

Peter Todd (OCAD)

Project Coordinator:

Jennie Ziemianin

The Mobile Experience Lab  
Ontario College of Art & Design  
<http://www.mobilelab.ca/portage/>

2008

## Introduction

Portage is an aesthetic, technical and social research and development project that had, as its objective, the installation of a Broad Locative Environment. The BLE was installed in an outdoor, streetscape setting and was interactively accessible to users of mobile devices. The elements allowed users to engage in various activities that are completed individually or in collaboration, including: playing musical instruments via mobile devices; viewing themselves as surveillance subjects on their phones; or setting off a series of visual and aural feedback loops based on the presence of mobile devices. The development and implementation of the BLE required a range of software, hardware and process innovations. These are detailed in the following pages.



The various components that comprise the mobile domain allow for a range of user interventions depending upon the sophistication of the user's apparatus and the user's access to networking. Portage looked at the relationship between this broad range of devices, communication networks and possibilities for unique social interactions.

To understand how we sought to intervene in this breadth, it is essential to understand the technologies Portage worked with. Our handsets predominantly included cell phones and a few PDAs. Cell phones, of course, have evolved rapidly to become portable computing and communication devices. Many now include a range of software formerly restricted to computers, per se, as well as hardware add-ons like cameras, MP3 players and GPS devices that allow them to be sold as multimedia-studios-in-your-pocket. We targeted

readily available high-end phones with all the latest features and low-end phones with only voice and text capabilities, while also testing on Blackberries. As well, we made possible the participation of non-cell phone users in Portage projects.

Our wireless communication protocols include those provided by carriers (e.g. GSM, GPRS and Blackberry service) and more autonomous services such as Bluetooth, Wi-Fi and WiMAX. Taken together these provide a reasonable range of content delivery options, though Bluetooth and Wi-Fi remain unintelligible to most users, even if their phones support them. The carriers in Canada offer services that are well behind those in other countries and have restrictive price plans for the few advanced features that are available, like data transfers. While third and fourth generation cell phone services are being rolled out or planned internationally, researchers and content innovators in Canada may have to find alternatives to the commercially available service providers in the short to medium term.

Given the parameters of the North American mobile environment, our content development plans took into consideration existing uses and familiarities, building on and extending them where appropriate. For example, by inviting potential users to send us a text message indicating their interest in solving a puzzle, we are able to capture their phone numbers and later send them voice messages that bring them further into the overall experience. Or instead of simply sending photos or videos to players, we send images they can control, or invite them to upload their own images as part of a collaborative, improvisational street performance. While users may be familiar with the concept of using their phones for viewing narrative content (downloading movies or TV has been a recent feature in cell phone advertisements), or playing screen based games, we have adopted content strategies that are outside of the game-story continuum.

Manuel Castells (2006), has suggested that mobility is the defining characteristic of mobile culture, arguing that users seek constant access while moving through spaces. In contrast, Ben Russell suggests that ‘locativity’ —the ability to connect and interact from within a very specific place—has attracted more attention from innovators in content

development and their advocates.(2004) Portage sought to embrace both biases, allowing new forms of connectivity in public, urban spaces that are stationary and mobile experiences. As such, we linked our mobile “street theatre” to a specific few blocks in the arts and entertainment district of downtown Toronto. This allowed us to investigate users’ preferences in mobile device capability (i.e. mobility, locativity, or other,) as well as novel forms of social interaction. To bring mobility and locative components to the overall user experience, we introduced additional features including: the material history, geography, architecture and culture of this specific street; and non-virtual elements that interacted with the otherwise mediated experience, including surveillance cameras, display screens and sculptural components. By encouraging participants in our production to become ‘actors’ we began to explore the function of agency in cultural experiences that are distinctly *mobile and locative*. Portage did not seek to offer political subject matter or ideology as focal points of installations but rather to encourage critical inquiry into a range of matters attendant to mobile society, including a host of false dualisms and assumptions regarding “normal” uses and users of mobile technologies.

We also provided mechanisms that both loosely and tightly linked content with mobile technologies. While some elements of our productions involved pre-packaged content that the user could manipulate (i.e. the basest form of interactivity), others contained no content whatsoever, creating instead mechanisms for users to contribute their own images and ideas.

## The Projects

The interactive and content delivery strategies revolved around several specific projects that were set up to interrogate the range of uses and development opportunities that were identified by the researchers and SME partners in the early stages of the project. These evolved, resulting in new and unexpected instantiations. Some techniques were refined as the projects moved forward, others were put aside in favour of more effective solutions. The projects, as initially envisioned are presented here, with detailed technical development notes.

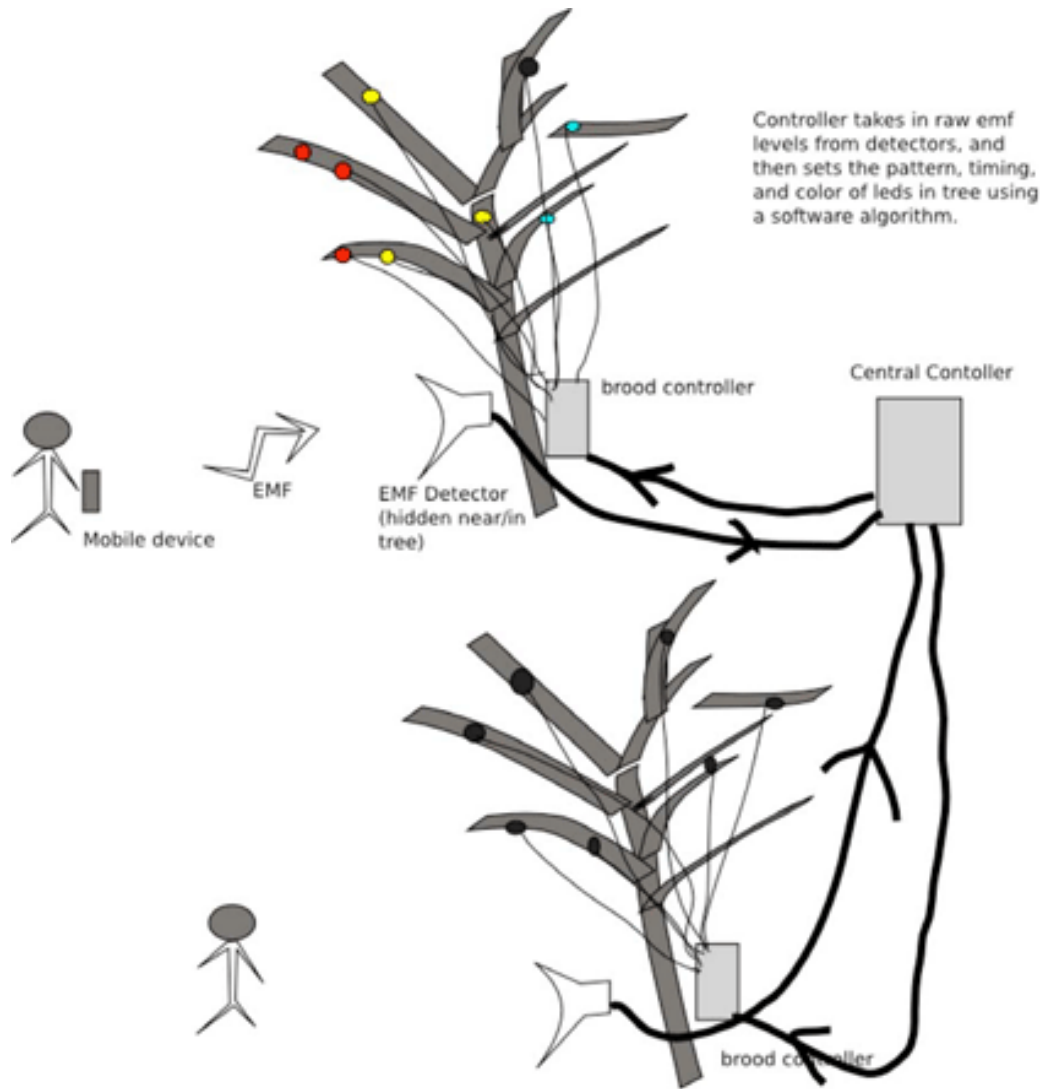


## Cicadas / EMF

The social theory of swarming combined with the technique of electromagnetic frequency (EMF) detection. Users talking on cell phones at certain locations along our target development area (John Street, a short street in downtown Toronto which is home to our lab as well as many media, entertainment and culture institutions) trigger swarms of virtual cicadas: sound and light emitting devices installed in trees. This draws parallels between signal emission as a communication imperative in both the human and insect worlds. It also creates a situation of passive interaction whereby users play a role in the experience without necessarily deciding to do so.



This project reflects the team's collective interest in users who don't see themselves as such, and pass through urban spaces as solitary users of PDAs, mobile phones or laptop computers. While members of youth culture or industry people might be engaging in social networking, mostly through social networking websites such as Facebook, in order to participate in social or business networks, most individuals in urban North America still consider their mobile devices as one-to-one communication devices. Cicadas seeks to remind passers-by that their mobile devices are actually small computers capable of a range of interactive experiences, and even able to trigger experiences without user intention.



### Design Steps:

1. Sensing Range: test sensing equipment in urban/park setting to determine range of effective detection (range of devices, distances, interference, etc.)
2. Mapping: determine optimum test areas based on high traffic and create sample measurements for prototype testing
3. Triggering: create hardware patch between sensor and central controller; interpret signal for detection strength

4. Displaying: create mechanisms to provide visual and/or audio feedback
5. Activating: create linkage between central controller and brood controller
6. Pattern Responses: develop algorithms to correspond with sensor data

### **Overall Architecture:**

1. User walks through urban/park setting and passively engages in sensor-driven feedback loop.
2. Multiple locations can be installed along a path, that cause distinct patterns to be displayed. User, if he/she becomes aware of the interaction, can cycle through the installations to create his/her own series of feedback loops.

### **Technical Development:**

Apr – Jun, 2007

- Did research into EMF sensors. Recommended purchasing the MicroAlert due to it's inexpensiveness and advertised sensitivity and sensitivity adjustments. Started building first version of Cicada hardware. Ran into some problems getting serial working on the PIC 18f2520. Will not communicate with host under any circumstance.
- Built a simple 12f683 based Nymph with a RGB led driven via PWM. Fixed 18f2520 problems, turns out that I had "Extended Instruction Set" mode enabled. Fixed and now using SDCC libuart library.
- Wrote an initial 3-wire master->slave networking protocol, "Primes" Currently just unidirectional communication. Successfully uses magic numbers for serial synchronization. The maser and slave are clocked within about 2% by the internal

RC oscillators of the PIC. The slave then repeatedly looks for a magic 32-bit number on the incoming data stream, upon seeing that number, reads the payload. The application has no need for error robustness, so this is a good enough, and very simple, protocol. Got above 2-wire protocol fully working with a single slave. Still no bi-directionality.

- Experimented with a 4-wire, V+, GND, CLK, DATA, Cicada protocol. Did research into I2C protocol and its derivatives. Didn't find much on extensions of it that can support nodes with randomized ids. The Cicada Nymphs will have to have randomized node ids to make installing easier, it's not practical to keep track of every one individually. Unfortunately practically all bus protocols are built with the assumption that nodes are individually numbered, the one exception being the Maxim 1-wire protocol.

Looked into games written in python for ideas, there is a pygames library and a \*very\* large amount of software written using it. Python has surprisingly sophisticated graphics abilities, such as fast OpenGL bindings. Started work on some basic frameworks to deal with nymphs and eventually present a behavior based API to people programming them. Some initial work on this in `io.pkt.client.portage.cicada.controller` (Peter Todd)

Oct – Dec, 2007

- The server app, "broodmaster", limits itself to being a "middle-man" layer. It passes data between the display and computation parts of the Cicada app. For display you get display targets, currently just a representation on the screen, but eventually the actual cicada hardware as well. The cicada behavior then gets implemented as its own Python module with the broodmaster software providing use full features such as which nodes are adjacent to other nodes.
- Got a "Black Hunter" emf detector designed for counter-surveillance use. Haven't done much testing yet, but with it on in a crowd, it seems to get "pings" from the cell phones at least every minute or so. Need to conduct more testing however.

- Made a mcp23017 i2c io expander based version of the Cicada hardware. No intelligence on the Cicadas, they were just simple leds. Couldn't scale design past 15 pwm output pins.
- Tried some more ideas for pwm for the io expander Cicada hardware. Definitely concluded that it's not a good idea. Research alternate possibilities, like dedicated LED controller chips like the PCA9632, a 4-bit i2c-based pwm controller. They will work, but sacrifice the ability to add sensors and change behavior. The cost is also not much less than a PIC chip.
- I've already implemented a master->slave system with PIC chips before for my 8<sup>2</sup>Automaton project. This included a nice bootloader to allow the slave nodes to be reprogrammed automatically from a single master. So using this architecture a new Cicada version was created, io.pkt.client.portage.cicada.nymph.16f818. The resulting board was based on thru-hole technology, except for the led, so that student labor in the charette would have a good chance of being able to help with the assembly.
- Turns out smd RGB leds are significantly cheaper than thru-hole. Cheap flat ribbon cable was used for the bus. Connectors are very quick and reliable to make up, saving a lot of time. The bus design left enough pins over to test two busses in parallel, an i2c control bus, and additional pins so that a 2 or 1-wire bus protocol could be tested on real hardware while still maintaining bootloader capability.
- Two commercial MicroAlert™ 2 radio/microwave alarms were used as emf detectors. While they worked fine in their intended purpose, reliably triggering when a GSM cell phone initiated a call within 5ft, hacking the circuit proved difficult.
- At first it was assumed that simply taking a signal off the peizo buzzer would work, but it turned out that the devices were so sensitive to signals that upon doing so they would never stop chirping. Any electrical connection what-so-ever would trigger this behavior. In the end a make-shift solution of attaching a condensor mic to the peizo buzzer was used. The condensor mic was then attached to an ADC pin on the PIC chip. Surprisingly, this actually worked.

- The io.pkt.art.8-2-automaton i2c bootloader reuse was a major success. With only about 2 hours work the code was ported over to the new hardware and enabling efficient development.
- Got cell phone activity detector working, based on design from <http://www.techlib.com/electronics/cellhelp.htm> and using tl072 op-amp. Works at a distance of about 1 foot. Only triggers at call start, seems to be same response as microalert, although with less sensitivity to noise.
- Worked on next generation design of the Cicada's, incorporating the op-amp based detection circuit. Switched to SMD parts to save space. Added a second led on the back to make orientation less of a factor. Added peizo buzzer.
- Put Cicada 2.0 on hold, read about a device called an rf log detector. This is a chip that when fed RF signals, outputs a voltage output proportional to the logarithmic db scale of the signal. Built a test pcb to try the lt5534 chip based on the example schematic in the datasheet. The biggest issue was really the size of the chip, it's SC70 package is the smallest pitch I've ever soldered. Fortunately, the lab has a binocular microscope, which proved very helpful. That said, I could not get the board to work, the output from the chip remained at a steady 3V regardless of what emf sources were present. (Peter Todd)

Jan – Mar, 2008

- Lots of testing of the rf log detector with various types of RF sources. Recorded outputted plots from oscilloscope and wrote up a report: <http://petertodd.org/tech/emf-detector/>
- Researched ethernet for microcontrollers for a new Cicada broodmaster. Started work on new rf log detector using Cicada design.
- Tried making use of solid-state thin-clients in the hope that they could also make for Cicada broodmaster boxes, no luck. The brand I had turned out to be locked down so that you can't get them to boot off anything. Tested out ways of interfacing the rf-log-detectors to the PIC microcontroller used. Turned out that the rf-log detectors, as designed, output pulses that can't be detected by the PIC.

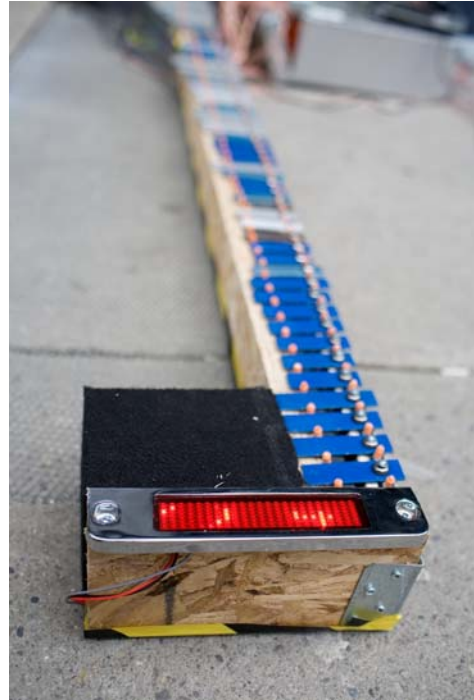
- Designed and tested hardware peak detection circuit for the rf-log detector. Re-did Cicada design to use this new circuit and sent out boards for manufacturing. Started designing ENC28J60-based ethernet controller board,  
io.pkt.client.portage.cicada.broodmaster.ethernet
- Assembled and tested new Cicada Nymph boards with emf sensors.
- Designed, built and tested new serial Broodmaster,  
io.pkt.client.portage.cicada.broodmaster.serial Built 25 more Cicada Nymphs with new emf detector circuit. (Peter Todd)

Another portion of the project involved detecting when the “cicada trap” box was opened, taking a picture on anyone who opened the box, sending a e-mail with this picture to the newspaper box project for display, and sending the photo to the opener's mobile phone if they sent a text message to our server with the body: “hero”. This project used the same techniques as used in Camera Hunter to detect the opening of the box and sending of the picture message. Because the camera in the box was the same model as the one used in Erasure Cycle, we experienced the same issue with the camera being over-exposed in a daylight environment, and implemented the same fix. (Rob King)

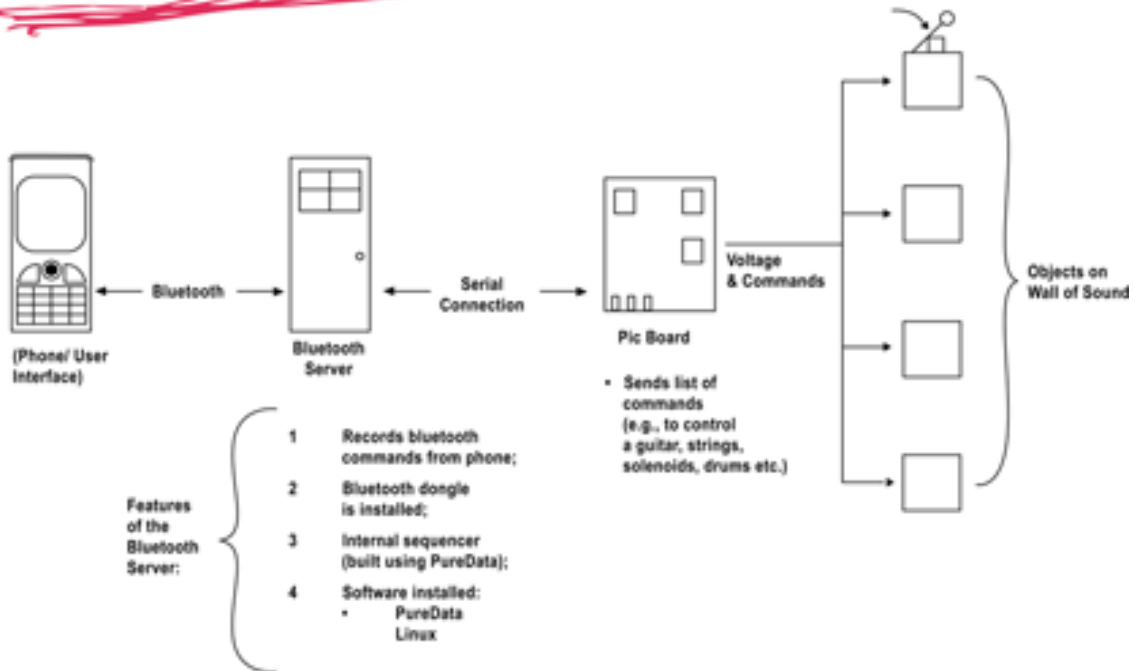
## Wall of Sound

Wall of Sound is an environmentally installed suite of mechanical, sound emitting 'instruments' that users can 'play' through an interface on their mobile phones. The opportunity to jam in real time with other users in the immediate vicinity creates a real world manifestation of mediated social networking.

This installation also begins to address social and cultural imbalances characterized as the mobile device: the diffusion gap inherent in class and economic uptake of new technology. Some of the instruments are available only to users with sophisticated devices (and the knowledge of how to use them), while other instruments can be controlled through simple phone calls or SMS messages. Certain instruments will also be installed that are meant to be played by users with no devices, by banging on a drum for example and introducing that signal as data into the system.



# WALL OF SOUND



## Design Steps:

1. Manual Action: Various percussion instruments are available on a sound sculpture that users can activate through physical manipulation.

2. Bluetooth Version:

Users download a Bluetooth application for their Bluetooth-enabled phone, which will feature a GUI by which they can issue commands to a Bluetooth server

3. EPBX Version:

Users without Bluetooth-enabled phones can issue commands to the wall by dialing into an EPBX server, which will process touchtone-based commands.

## Overall Architecture for Mobile-Activated Use:

1. User chooses an object on the wall using their mobile device (using a Bluetooth application downloaded from the wall, or by dialing into the EPBX server)
2. User 'plays' the chosen object using their mobile device (using a GUI interface for Bluetooth version, or by using the device's buttons for the EPBX version)
3. User's actions are translated into commands that are then sent to the control server (which contains both a Bluetooth Server and an EPBX server). The server processes the commands using a PUREDATA application and deploys instructions to the PIC hardware via a serial connection
4. PIC hardware sends commands to physically control/affect the chosen object on the wall. Example: if a user programs a sequence of beats on their mobile device to control a drum instrument on the wall, the PIC board would turn a solenoid on and off, causing it to strike the drum in the rhythm pattern specified by the user.

### **Technical Development:**

Apr – Jun, 2007

Built a pure data based demo to control a PIC-based hardware solenoid control board. The PIC side of things was a minor affair and can be done in many different ways. Need to investigate what existing hardware can provide this functionality, such as MIDI stuff. Pure data as a drum machine is surprisingly tricky to set up; unsure if the interface will meet the goal of giving non-programmers any hope of modifying its functionality. (Peter Todd)

Evaluated some open source VOIP libraries; no clear winners, but pjsip was successfully used for a demo. To do will be to investigate interfacing pure data to VOIP. (Peter Todd)

Finished PJSIP VOIP demo. Created Wall-of-Sound controller board to drive relays on Wall of Sound. Finished Wall-of-Sound control software. Implements a drum machine in pure data using tables. Communication to drum hardware via serial link, communication from phone via the bluetooth server using a simple FIFO and socat to take a TCP stream and get it to a simple custom pure data patch in the simplest way. Pure Data turned out to be surprisingly inflexible. Getting the table read stuff working for the drum machine was very tricky, far more difficult than simply writing it all in a high level language like Python. (Peter Todd)

Oct, 2007

We created a mobile device interface (using Mobile Processing), to control our PureData-based beat sequencer, which would in turn play the Wall of Sound physical prototype (a bank of solenoids) according to the rhythm set by the user. We decided to build two versions – a bluetooth version for higher-end phones, and a web-based version (which would access the sequencer via GPRS) for maximum compatibility. Since our Wall of Sound prototype contained 8 solenoids, we built an initial interface which was basically an 8x8 grid of checkboxes – 8 beat slots for each solenoid, which was sufficient for testing (to clearly discern a difference in rhythm when it was changed). Each time a box was checked/ unchecked, data would be sent to the server:

*1) Bluetooth Version:*

Built a relay app (in Processing) on the server machine (Macbook Pro), which would receive data from the mobile phone via Bluetooth, and forward it to PureData server (IBM Laptop). The relay app was equipped with an open serial port, so that the mobile phone interface could send rhythm data to it via Bluetooth.

*2) Web (GPRS) Version:*

An identical interface was built for the Web version, but which sent data via HTTP instead. Rob built a relay app in Python (on a Linux server), which would receive the data and forward it to the PureData server.

The server-based Processing app would crash every time the Puredata application became unavailable – we need to incorporate failsafe code which would enable the app to go idle if connection is not found, and try periodically to connect instead of throwing an error.  
(Ken Leung)

We set out to create a Wall of Sound prototype app that would analyze a user's speech, and then create a MIDI musical piece mimicking the contained pitch/tempo/rhythm variations. To make the application accessible to the widest variety of users, the speech data would be sent to the Wall simply by calling in to a local telephone number (no bluetooth or wifi capabilities necessary). For the prototype, we set up the following components:

- a VOIP call-in account with Gizmo ( [www.gizmoproject.com](http://www.gizmoproject.com) ), which would enable a mobile phone to call directly into our server via a local phone number.
- A Gizmo client on our server which would receive voip calls to the call-in number
- Audio-to-midi translation software, installed on our server.

Using the above components, we were able to successfully implement a prototype that would convert all incoming sounds from the voip connection to MIDI sequences. We experimented with piano, beat tracks, and a number of other instruments, all of which worked quite well. (Ken Leung)

A prototype of this interaction was developed using a Gizmo VOIP account which was then connected to an application that would translate the pitches that were whistled into MIDI messages (low fidelity signals that abstract musical data into discrete messages such as “play the note middle C, now play F#”). These MIDI messages could then be sent to digital synthesizers, or be used as control messages to trigger physical instruments.  
(Rob King)

Jan – Mar, 2008

This project consisted of four components: a set of three motorized kick-drum pedals with cow-bells mounted on them which could be sequenced using a Bluetooth application on a mobile phone, a ventilation grate with a piezo disc mounted to it which could sense vibrations and trigger sound-clips, and a glockenspiel with 36 solenoids mounted below it's keys which could be triggered either by calling into a phone number and singing into it, or pressing the keys on a mobile phone keypad.

The set of three drum-pedals was controlled by an Arduino microcontroller connected to a simple relay circuit. To control the drums we built an 8-step sequencer in Mobile Processing, which used Bluetooth to communicate to a Python script on a server in the field which would then trigger each drum according to the pattern input by the user. While this system worked well in the lab, two issues made it problematic in the field. Firstly it was found to be difficult to deal with distribution of the sequencer application to mobile phones. Secondly it was discovered that on the street the Bluetooth connection was only reliable up to a distance of about 2 meters, after which performance became intermittent. To deal with these issues a second version was developed which allowed users to connect using the web-browser on their mobile phone. While this version is not free to use due to carrier data fees (except on WiFi enabled phones), it has proven to be significantly more reliable.

The implementation of the sound-triggering ventilation grate proved to be relatively simple, using a piezo contact microphone the vibrations of a grate were monitored by doing simple audio level analysis in a Processing application. The one issue that appeared on the day of our beta test was that the ambient vibration of the grate would fluctuate over time sometimes falsely triggering the sound samples. To accommodate this issue, the triggering algorithm was changed from a simple threshold triggering to a more adaptive algorithm that would trigger when the momentary audio level raised above the moving average of the last 200 audio-level measurements plus a threshold value. Thus the triggering of sounds could adapt to changing ambient noise levels.

The implementation of cell phone keypad triggering for the glockenspiel was relatively simple. Using the same Asterisk server as used for the Erasure Cycle, a Python script was written that, when called, would play a message asking users to press the numbers on their phone keypad to play the glockenspiel. After a user input a sequence of numbers using their keypad, these numbers would be transmitted over the internet to a server at the installation site and keys corresponding to each number would be played in order. This sequence would be looped three times.

Implementing the pitch detection component of the glockenspiel proved to be much more complex. The first issue was getting access to the audio data of an incoming VOIP call. Unfortunately it proved to be difficult to access this data using Asterisk so we had to resort to an alternate solution. We ended up using the Gizmo VOIP client set to auto-answer with a hardware sound loopback which fed the incoming audio into a custom Processing application. This application used the Fast-Fourier Transform to discover the monophonic pitch value of the incoming audio signal. This value was sent over the internet to a server at the installation site where it was then used to trigger a note on the glockenspiel. While functionally of this interaction worked it had some aesthetic issues. Firstly, the glockenspiel didn't have a full scale of notes but instead the range of notes were mapped to a musical key. Because of this the mapping between vocal pitch and glockenspiel pitch appeared to be somewhat arbitrary. Secondly the mapping of a wind instrument (the human voice) to a percussion instrument (the glockenspiel) proved difficult as the only way to represent a prolonged note was to trigger a single key on the glockenspiel several times. After testing it was decided that the phone keypad control of the glockenspiel produced better results.

Once again on the day of the beta test we found once again that OneZone WiFi could not be relied upon for connectivity in the field. Even though all the high bandwidth VOIP connections were being handled by a server in the lab and only low-bandwidth control data was being sent across the internet to our on-site machines, changes in the OneZone network firewall or proxy configuration made even this low bandwidth connection

problematic. Fortunately it was possible to switch from WiFi to WiMax in the field for much better results. (Rob King)

Feb 2008

*Sidewalk Grate Component:*

- With Rob King, created application in Processing ([www.processing.org](http://www.processing.org)) to receive audio input from piezo microphone
- Used ESS library to process incoming audio, and calculate the audio level
- Experimented with thresholds and wrote code to trigger playback of random voice files (recorded by Jennie Ziemianin) when levels exceeded set limits (Ken Leung)

Mar 2008

*Drum Machine Component:*

Modified Beat Sequencer code for mobile phone (from early bluetooth-enabled solenoid instrument prototype) to work with Drum machine:

- Reduced number of instruments on interface to 3 (3x10 grid)
- Added code to receive current state of the Drum Machine (which drums are currently playing, and the current rhythm sequence), on startup
- Modified interface and Bluetooth connection code to reduce the number of steps to connect to the Drum Machine – user no longer needs to choose the target machine, connection is automatic on startup

Encountered various Bluetooth connectivity problems using test Nokia N70 phone in the lab environment – turned out to be a phone issue, as connectivity was near perfect when using Nokia 6131 instead. (Ken Leung)

*Drum Machine Component:*

More drum machine work, building hardware and getting the motors to work. Started working on Glockenspiel controller software, `io.pkt.client.portage.glockenspiel`. Helped

build up the Glockenspiel. Glockenspiel first stage working with first version of hardware, finds the dominant frequency in an audio input, and triggers solenoids in response. More work on drum machine, made a simple sequencer in Python with hardware output.

Built controller board and associated software for full, 40 solenoid Glockenspiel. Wired up the full 40 note Glockenspiel and associated driver board and tested hardware with debugging library. (Peter Todd)

## **Wall of Poetry**

Apr – Jun, 2007

Built a mobile phone interface for Iteration 1 of the Wall of Poetry component of the Wall of Sound project (Bluetooth Control + Canned Audio Content), using Mobile Processing. All words in the canned poem were stored in an array in the application, and then displayed in list form on the GUI. The order of this list could be rearranged via the GUI – the arrow keys were used to scroll through the list (the current word being highlighted), and pressing the ‘2’ or ‘8’ number keys would move the position of the selected word forward or backward in the list. (Ken Leung)

Oct – Nov, 2007

### *Mobile Interface Component:*

Added new capabilities to the existing interface, enabling it to send the current arrangement of the poem (stored in an array) via Bluetooth to the Poetry Wall server component.

First, I added a Bluetooth device search mechanism to the interface application, which would first build a list of all Bluetooth devices in the vicinity, and then extract from this a sublist of all Bluetooth devices with open serial ports (which I found to be the best way to send data to the server-based component). The user would then be prompted to choose from a list of the serial ports found (a step included for debugging purposes – in the final version, the app should choose Poetry Wall server automatically).

A send feature was also added to the existing interface, so that once the user had rearranged the poem to their satisfaction, they could send it to the Poetry Wall by clicking on the ‘9’ key. An exit button (\*) was also added -- the ‘\*’ made most sense for this since it was the furthest distance from the user’s thumb, and therefore minimize accidental exit.

Once the ‘9’ key is pressed, the application established a serial connection to the server, and transferred word-order data for the poem. In my first attempt, I sent the data in its original form, which resulted in it becoming garbled by the time it reached the server (since the application converted data to ASCII in transit). The solution was to convert the data to ASCII prior to sending (which would be converted back to numeric format on the server)

Used Nokia N70 and Nokia N95 as test devices.

## *2) Poetry Wall Server Component:*

Created a server-based application which would receive poem word-order data from the mobile device, and then play an audio version of the newly ordered poem, as well as display it visually. I built the application using Processing ([www.processing.org](http://www.processing.org)), and the additional Serial library (<http://www.processing.org/reference/libraries/serial/>) facilitate Bluetooth transfer. I experimented with several audio libraries (Sonia, ESS, Minim), and decided on Minim (<http://code.compartmental.net/tools/minim/>)

due to its ease of use for basic audio playback. The audio for each word in the poem was prerecorded (since this iteration involves a canned poem), and stored as wav files on the server. For this prototype, a Macbook Pro was used as a testbed.

On startup, the application opens a Bluetooth serial port, which listens for external Bluetooth connections (on the Macbook Pro test system, the serial port had first to be enabled in Network Preferences to function properly). Once poem word-order data is received by the server (from a mobile device), the data is converted to ASCII, and then stored in an array. The app would then cycle through each word in the array, each time playing the corresponding audio file. For the Minim library, it was necessary to enclose the audio playing code in a while() loop for playback of entire clip to occur – until I did this, only small portions of each audio file were being played before jumping abruptly to the next file.

Words in the poem were also redisplayed on the screen (in their new order) each time a new poem was ‘sent’ to the server from a user. (Ken Leung)

## I Spy / Camera Hunter / Erasure Cycle

A twist on the roles of surveillance (and its citizen-based inversion, sousveillance) in urban cultures. Specially installed Wi-Fi cameras transmit images to large display screens at street level and passersby are invited to

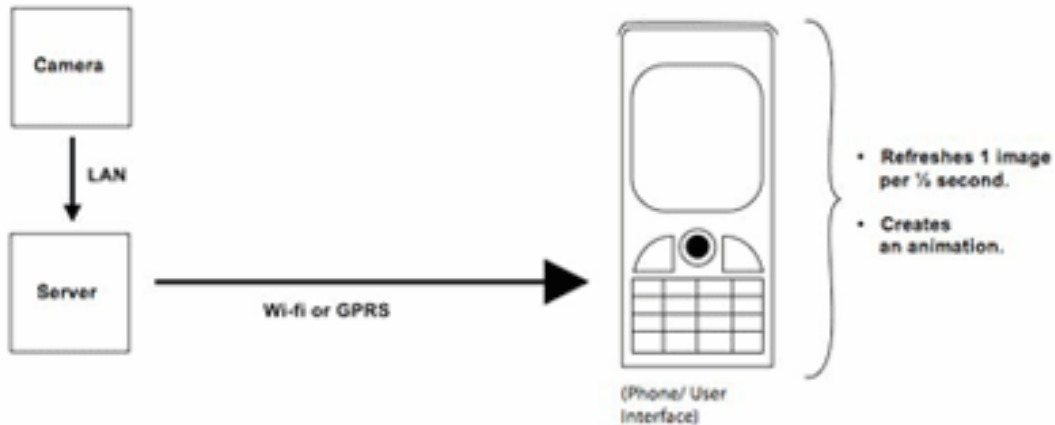


control, manipulate and record the images of themselves (either for future users to stumble upon or as a take-away downloaded to their phones). The always-on nature of the surveillance apparatus provides a further introduction to users to move from passive to active participants. In this case they are able to combine the image of themselves with images of other locations along the street through keystrokes and the actual movement of their bodies within the frame. Current images can be blended with past images, highlighting for example the contrast and tension between daytime use of this street and its transformation after hours into the most tensely packed nightclub district in North America.

# I-Spy

## ITERATION 1:

Sending still photos of self to phone.



In rapid prototyping sessions, or charettes, we erased the thematic of surveillance and highlighted the technological possibilities. By removing the context of surveillance – which our project team was deeply committed to understanding in relationship to social surveillance context of city life and users of mobile phones—our team (led by designers engineering on open source platforms) created an imaginative and playful re-purposing of the surveillance camera as one leg in a system of linked screens, cameras and phones. By playing with these items, we came to understand surveillance cameras as something that can be reimagined as other types of devices with roles that differ from the standard practices of and dynamics produced by photographic cameras.

In one instance, for example, we, critically, turned the camera on itself, making it a tool that allows one to consciously insert herself into a social narrative of her own making. More, we repurposed the cameras to overlay or fill the surveilled image. As such, users approaching surveillance cameras can control – not the angle, or focal point of the camera but instead, the actual image. As such, the surveilled subject is no longer subjected in the usual case of photographic images. This image, instead, can be entirely distorted by the image of the user approaching the camera, making the surveillance

camera something that subjects the conscious user, by choice, rather than the unsuspecting passerby who is unintentionally caught on camera. The surveilled subject views this manipulation either through the public screen or via her cellphone.

What is surprising and exciting about this new technology is not that erasing the context creates innovation. Rather, the inherent bias of the technology – that surveillance cameras surveil – was not eliminated. Rather, it is maintained and repurposed in a manner that reveals and critiques the notion of surveillance. This illustrates the difference “between the practice of data processing and its narrative”. (Castells) Phenomenologically there is something very interesting that occurs here, because the experience is inherently both playful and irreverent. To erase another subject with one’s self, and to supplant pieces of their image with one’s self is an act of power – that is simultaneously playful and teasing, even as it is willful and manipulative.

More importantly, this experience occurs in a different context that is both mobile and urban. Were game producers to place screens and cameras in the street for play, the experience would be akin to a simple movie set, where the film created was live and interactive, and where camera movement and control is replaced by users who manipulate the image via the movement of their bodies through space. While anyone in the public space can view the experience on the public screen, the mobile user, who is actually surveilled can view herself as the subject of her phone, as she moves in real time, or on the public screen, as the subject of what can be termed a public film.

### **Overall Architecture:**

Network-enabled camera streams captured video footage constantly to control server

Server performs post-processing on video stream (e.g. image blending and filtering), and sends it to user’s mobile device

User receives video on his mobile device (which is updated in real-time)

### **Communication Schemes:**

Wifi Version: Users with Wifi phones can access the video streams using Wifi connections to Ontario Hydro One-Zone hotspots

GPRS Version: Users with non-Wifi phones can retrieve the video streams over their cellular network via GPRS

### **Types of Video Streaming:**

Still Image Sequences: Server sends camera footage to phone using a constantly updated sequence of still images (result will resemble a flip-book)

Real-time video: Real-time video streaming using a Java or Processing interface, via RTSP protocol. A streaming server is required for this configuration

### **Types of Image Post-Processing:**

Image filters:

Desaturation (color to black & white)

Image Grain

Color Adjustments/ Manipulation

Blending two videos/image by varying alpha levels (transparency)

Isolating components of one video (e.g. a human form) and inserting them into a second video

## **Technologies/ tools to be used for image processing:**

ImageMagick, NetBPM, Processing

## **Technical Development:**

Apr – Jun, 2007

Built a prototype of the I-Spy application, using WiFi as delivery method, and real-time updated still images to stream video. Mobile device connects to a webpage on the Apache web server via built-in web browser, which sends a continuous stream of still images every 0.5 seconds. I used the Nokia N95 (a WiFi-capable phone) for testing. Image refreshing is still a bit rough, due to variations in WiFi connection strength and file size of the sent images. (Ken Leung)

- Built additional iterations of the existing I-Spy prototype , which would perform image-processing on raw camera footage to place the user within a simulated environment (e.g. within a historical setting).
- The first image-processing iteration utilized a PHP page (which would refresh with a new frame of footage from the network camera every 0.5 second), and the JMagick module (an ImageMagick implementation for PHP). However, JMagick proved fairly limited in its capabilities, which led to the next iteration, which used a native ImageMagick installation instead.
- The second iteration consisted of both a frontend process (XHTML page which would refresh with new frames every 0.5 seconds), and a backend process (which retrieved and processed images from the network camera). The backend script would execute WGET continuously to download frames of footage (in JPEG format) to the server. As each frame is downloaded, the script would execute ImageMagick commands (to add desired visual effects), and then mark the frame as the current image (by saving it as 'current.jpg'). On the frontend, the XHTML

page (which is accessed by the user on their phone) would grab and display a new version of the current image ('current.jpg') every 0.5 seconds, thereby creating a continuous animation.

- Using the mechanism from the second iteration, I created a new prototype which would place the user virtually within a historical scene. ImageMagick color filters were used to transform the camera images into grayscale. Using alpha blending filters, the grayscaled camera image was then combined with an existing historical image. Finally, Imagemagick's random noise generator was used to add image grain, to further reinforce the illusion of an aged film. (Ken Leung)

Oct, 2007

was the creation of frameworks and experiences for the ISpy project. The first step in this effort was finding a means to access the data from two AXIS network cameras and stream it to a mobile phone. The first method of streaming this video data took advantage of the built in 3GP encoder in one of the network cameras. Using the WiFi connection of the N95 phone, it was found to be possible to stream high quality video to the RealPlayer application on the phone. Unfortunately, though the video was of excellent quality there was little that could be done with the stream aside from view it. That is, no manipulation of the stream was possible. Additionally it was decided that the bandwidth requirements could be prohibitive to anyone accessing the stream over the cellular network.

Building upon the Portage team's previous findings that it was possible to create a low bandwidth, low-frame-rate stream of still images using the HTML META REFRESH tag, I developed a server in Python that would grab still image captures from the network cameras, perform various manipulations on the images, and stream the manipulated image stream to a cellphone. One of the issues found in the original version of this solution was that every time the image refreshed (about once a second), the cell phone screen would go blank, and take a moment to load the new image. This created a flickering effect that was not entirely pleasant. Thus a second iteration of this system was developed which used JavaScript in the page that was accessed by the phone to refresh

only the image rather than the entire page. This solution gave the desired effect of a flicker-free image stream.

With this new framework in place, it became possible to explore the experiences made possible with a low-bandwidth live image stream. During the charette we developed one such experience, playing with the idea of inserting historical imagery into the live surveillance stream. In this iteration, the server software would keep an archive of all of the images captured since it started running. Users that accessed the server on their cell phone would be presented with a live stream of the view of the network camera. This stream would occasionally be interrupted by inserted imagery either faded from a previous time since the server began, or by archival imagery taken from the same location. (Rob King)

#### *Superimposition of Historical Imagery:*

We experimented briefly with VVVV (a software for manipulating realtime video). Since we decided earlier to concentrate on the still-image-sequence solution, we instead built a Python application, which used PyMagick (a Python implementation of ImageMagick), which superimposed an existing image with the current feed by calculating pixel/brightness differences between the two images, and feathered the superimposed image to facilitate blending. This yielded an unexpected effect – a ghostly halo around the objects in the superimposed image, which strengthened to its ‘historical’ aspect.

#### *Display of Archived Video Feeds:*

We created an interface below the video feed on the phone screen, labeled ‘Now & Then’. Clicking on ‘Then’ would reset the video feed to the beginning (i.e. rewind the realtime video to the first frame that was captured by the camera), and play the frames forward until it reached the ‘present’ yet again. Clicking on ‘Now’ would return the video instantly to the current frame. (Ken Leung)

Nov, 2007

Building on the success of the prototype of ISpy developed for the charette, much of November was focused on developing a variety of iterations of possible experiences built upon the ISpy server. The open-source Ubuntu operating system was installed on one of the Portage computers and set up as a development server for the iterations. By setting up the server to have a static IP address accessible by the internet outside of our lab we were able to give access to the developed experiences through the Toronto Hydro Telecom Wi-Fi system.

Following the theme of historical image insertion work was done to develop a film grain effect that could be applied in real-time to the image stream. This was done using the open-source Python Image Library which was adapted to provide real-time compositing capabilities. By randomly superimposing a series of images of film grain and dust on-top of the live image stream, and reducing the image saturation a reasonable impression of aged film could be achieved. This technique was applied both to live footage as well as historical imagery to achieve a visual consistency when mixing between the two sorts of footage. Similar techniques could be used to produce a number of different effects.

With the new image processing capabilities given by the ISpy server we were able to develop another iteration that used motion tracking algorithms to detect what areas of a space are occupied by participants. In this iteration, an image of one of the lab members was superimposed upon the live footage. Users interacting with this experience would see that there were figures in the space they see on their phone that don't exist in real life. If the participant attempts to move to interact with this virtual figure, then the figure would jump to another location in the space.

This iteration was found to be an interesting interaction so we decided to further explore the potential of such augmented reality experiences. For the next iteration we decided to play with perspective by virtually extending the environment of the Portage lab using the perspective of a second network camera. The ISpy server software was enhanced to capture the streams from two different network cameras, and then using a pre-generated alpha mask, composite the image stream from the second camera upon a portion of the

wall in the first image stream. This created the impression that there was another room adjoining the lab that is only visible when viewed on the cellphone. This also created an interesting effect as users moved around the space, as there would be two instances of the participant, one from the normal view of the first network camera, and a second one in the virtual adjoining space visible from another perspective.

One problem experienced in the first attempt at this iteration, was that when the participant moved in front of the portal to the virtual space, the space would be superimposed on top of the participant, thus destroying the illusion. Thus an effort was made to find a method to extract the image of the participant from the background, and superimpose the participant on top of the virtual space as well. Using some of the techniques developed in the motion-detection iteration, it was found to be possible to extract the participant from the background by comparing the current image of the space with a reference image of the empty space. The results of this comparison could then be used as a mask to cut-out the image of the participant from the background and superimpose that image upon the composite of the two network cameras. Generally this technique was found to be quite successful, though changes in lighting conditions can severely impact the image fidelity.

Following the development of the technology necessary to isolate participants from their surroundings, a few iterations of ISpy were developed to take advantage of this capability. One of the simplest iterations used masking to replace the floor of the lab with a static image of a lawn. Thus participants could see themselves occupying imaginary spaces through their cellphones. Another iteration completely replaced the background with an image of the sky, and replaced the silhouette of participants with a grassy image, creating human shaped islands in the sky. Finally another iteration allowed users to capture and superimpose images of their silhouette, allowing users to create interactive paintings using their bodies. (Rob King)

Concurrent with the work developing iterations of the ISpy project, I was also working on finding a method to stream full motion video (15FPS rather than the 1FPS of the

image streaming technique) to a cellphone while maintaining the capability to manipulate the content of the video. The typical method of doing this would require licensing the Real Networks *Helix Server Unlimited*, the only Real Networks server that supports streaming to mobile devices. This product is normally only available to enterprise customers though, and so I began searching for a free/open-source method of doing this. I investigated both the open-source version of the Helix Server, and another open-source streaming server called Catra. Neither of these systems were found to be adequate for our requirements.

Eventually I found a solution using a combination of Video LAN Client (VLC) and Apple's open-source Darwin Streaming Server. In this solution a video capture card (a Hauppauge WinTV card) was installed in the server machine. The VGA output of the server was run through a scan converter to convert the VGA signal to a composite video signal, and fed back into the video capture card. VLC was then used to capture the video signal from the video capture card and feed it into Darwin Streaming Server. Cell phones can then connect to a MPEG 4 video stream from the Darwin Streaming Server using RealPlayer. This method provides a versatile solution which allows nearly any computer to be connected to our streaming server and output a full motion MPEG 4 video to a cell phone. The two limitations that we found with this system are the increased bandwidth requirements of a full motion video stream, and latency between what is fed into our streaming server and what is seen on the phone. Currently there is a unavoidable 5-10 second buffering delay in the system. While this is acceptable for many applications, it means that if a user is viewing a live stream of themselves, any movements they make will be reflected on the cellphone screen 5-10 seconds later. This can be a disconcerting effect in interactive video pieces. (Rob King)

Dec, 2007

One of the elements that was considered for incorporation into the ISpy project was some sort of positional aspect. At the moment though very few phones have GPS capabilities.

As such during a period in December, research was conducted with the goal of finding a more readily available means of finding a participant's general position along the John St. corridor. The main method I began research into was developing a map of the Toronto Hydro Telecom Wi-Fi access points along John St. and correlating the IP address of the hot-spot with its physical location. Using the KisMac Wi-Fi scanning software in combination with a Bluetooth GPS I walked down John St. collecting data about the position, MAC address and signal strength of every Wi-Fi access point. In total data was collected on 470 access points, 52 of which were found to be Toronto Hydro access points. Upon analysis of this data however, it was found that the Toronto Hydro Telecom network works on a mesh networking system which allows a number of access points to share a single IP address, thus this method for user positioning would not work in this situation. (Rob King)

For the Ispy projects, we were considering using the OneZone Wifi hotspots installed on John Street by Toronto Hydro as locative beacons. This would enable users with wifi-enabled devices to trigger game experiences based on location, without the aid of an external GPS device. Rob King and Ken Leung started by ascertaining the exact locations and SSID's of all Onezone hotspots along John Street. To do this, we ran KisMac (a software which sniffs for wifi access points) on a laptop equipped with a GPS device, while physically walking the full length of John Street. This yielded a KML file containing a list of all hotspots and their associated GPS coordinates, which we then opened in GoogleMaps to build a map of John Street with locations of all Onezone hotspots marked. During the walk, we also continuously accessed a streaming video via a wifi-enabled phone (connected to the OneZone network), to test for any wifi 'dead zones' between hotspots. Based on the data, we discovered that all Onezone hotspots had the same SSID, which could explain why we found wifi deadzones at consistent points between hotspots. (Ken Leung)

Jan – Mar, 2008

The follow-up project, with the working title “Hunter or Hunted”, involved placing cameras in various store-front windows. By sending a SMS message with the text “hunter” in it users would receive a live image or video from one of the cameras and be prompted to explore John St. and try and blind the cameras by covering them with their hand. The first portion of this which was developed was a simple algorithm for telling when a camera was blinded or not. The previous experiments in the lab using image processing and motion detection proved to be somewhat delicate when faced with changing light conditions. Because this installation was designed for the street a greatly simplified system was developed to adapt for constantly changing light. The basic algorithm to detect if a camera was blinded was as follows: The camera takes an image and converts the image to black and white. The brightest and darkest pixels in the image are compared. If the brightest and darkest pixels are close to the same value (as determined by a threshold) then it is assumed that the camera is blinded.

To send the user images or video from the camera to their phone, we originally intended to use the JavaScript refresh/browser-based low frame-rate video streaming technique we developed for iSpy. It soon became apparent however that this system would prove to be prohibitively expensive on non-WiFi equipped mobile phones as data-rates on Canadian carriers are high. As an alternative to this method for the base interaction with this project we decided to test MMS (picture messaging) as a content distribution system. To test this interaction we developed a SMS receiving and SMS/MMS sending server in Python. We used the Python MMS library (<http://python-mms.sourceforge.net/>) to compose MMS messages using images taken from our network cameras. The compiled MMS message files were then uploaded to the lab's web server, and a bulk SMS provider (<http://www.clickatell.com/>) was used to send MMS notification messages to user's cell phones. This approach proved to be significantly cheaper to the end user. To receive SMS messages we used a Nokia Series 60 mobile phone paired to a server computer using Bluetooth, and running a customized Python program to forward any messages over the Bluetooth serial link.

One of the major issues in the preparation of this project was finding a way to connect the network cameras to the internet from their sites along John St. Originally we intended to use the Toronto Hydro OneZone WiFi Network. There proved to be several issues with this solution however. First and foremost the network cameras were unable to connect to the network because OneZone has a web-based authentication system and the cameras have no browsing capabilities to allow them to log-on. To work around this issue we considered using computers at each site to log in to the network. This proved to have it's own issues though as the One Zone network only provides private IP addresses behind a single public IP address. Because the network cameras we used work on a "pull" model (where any client devices request content from the camera), they proved to be inaccessible when masked behind the single OneZone IP address. Finally it was decided that we would use Bell's WiMax network to connect the cameras. This solution gave each camera a public IP address accessible both from the server residing in the lab, and on any mobile phones on the street. This solution also provided a means to stream live video from the cameras to WiFi equipped cellphones. The network cameras we were using are capable of producing a 3GP video stream, which is viewable in RealPlayer on several mobile phones. Because each camera had it's own public IP address on the Bell WiMax network, users were able to access the live full motion video from each camera on their mobile phones.

The alpha test of this project on March 14th unveiled a few significant issues with it's original implementation which made it essentially unusable. Because the cameras were not placed directly against the windows in the shops when users attempted to cover them from the outside of the window, the reflection of the shop interior was picked up by the camera and the server failed to detect that the camera was blinded. This was compounded by the fact that the cameras had an auto-gain function enabled which made it so that unless a camera's lens was completely blocked the camera would always have an image.

To accommodate multiple users attempting to blind three cameras in the field a queue system was implemented. When a user sends the server their original "Hunter" message

the user is entered into a queue of hunters and assigned a camera. The camera which they are assigned is marked as “In Use” and cannot be assigned as a target to other users until it is blinded. Because users in the alpha test were unable to actually blind the cameras, after the first three users connected, new users would not receive a target. To fix this issue for the beta, a 10 minute time limit was added to the game so that if a user could not find a camera within 10 minutes they would be removed from the queue and the target camera would be assigned to another user in the queue.

In preparation for the beta test the auto-gain function of the cameras was turned off and the cameras were positioned flush against the windows. These modifications significantly increased the success of the project. (Rob King)

“Erasure Cycle” utilized the motion detection techniques developed in February to interface with mobile phones and physical devices. In a bicycle shop window a large LCD screen, a web-camera, and a motorized bicycle wheel were set-up. Passers-by were invited to ride the “No-Touch Bicycle” by waving their hands in front of the screen to rotate a set of virtual bicycle wheels in the corners of the LCD screen, as well as the physical motorized bicycle wheel in the window. As users made the wheels rotate a bicycle icon on the screen would progress across a track towards a “finish line”. Once the users move the bicycle to the finish line it is revealed that pictures have been taken of them in funny positions while they tried to rotate the bicycle wheels. By calling an on screen number, the user is given the choice to delete the images, or have them sent to their mobile phone in a picture message.

The first step in developing this project was to find a suitable framework to develop the user interface on. The iSpy projects used a combination of the Python SDL game library binding PyGame and the Python Imaging Library to do user interface and motion analysis. Because of the visual effects we intended to develop for this project it was decided that we should use the 3D OpenGL library within PyGame for it's superior performance and visual capabilities. To make this change a simple 3D scene-graph engine was implemented. Because the iSpy projects used network cameras for image

analysis, it was easy to use the Python Imaging Library to do image processing as capturing an image simply involved downloading it from an URL. The Erasure Cycle project however required full motion video for it to be a fulfilling user experience, and as such we had to switch to Intel's open-source computer vision library: OpenCV for video capturing and image processing. The algorithm used for motion detection was relatively simple: each video frame was split into a 3x3 grid and compared with the previous frame. The total difference between the two frames was then summed for each section. The resultant values give a reasonable approximation of the amount of motion in each section of a frame, and because the differences are measured frame-to-frame changing light conditions have no significant effect on this technique.

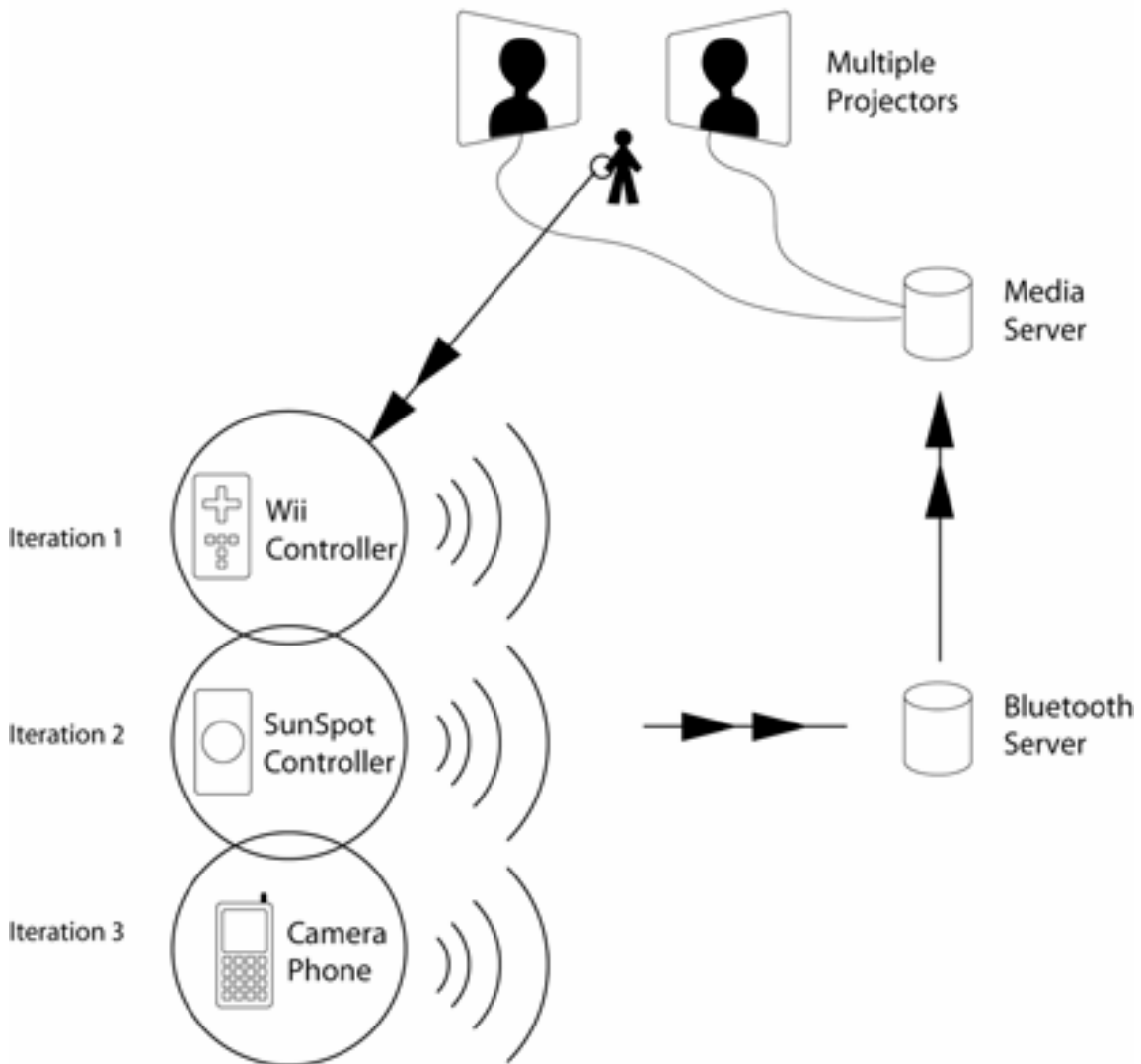
The second part of this project to be developed was interfacing the computer with the motorized wheel. To do this we used an Arduino microcontroller. The Arduino was loaded with the Firmata firmware (<http://www.arduino.cc/playground/Interfacing/Firmata>) which then allowed the Arduino to be simply controlled from Python using the Pyduino library (<http://orphans.atspace.com/pyduino-0.1.tar.gz>). The Arduino was connected to a relay which controlled the motion of the motor connected to the bicycle wheel.

Interfacing with the mobile phone was done using Asterisk, an open-source Voice Over IP based telephony server. To send the MMS messages when requested, the same bulk SMS service provider as in the Camera Hunter project was used.

In the alpha test it was found that because of the difference in brightness between the indoor light of the lab, and the daylight of the test location the camera we were using required several layers of darkening filters to be placed over it's lens, otherwise the image would be overexposed. It was also found that the internet connection from OneZone was not reliable enough for Voice Over IP connections, and as a result the mobile portions of this project only worked about 50% of the time. For the beta the Asterisk server was hosted in-lab, and only a very low-bandwidth data connection was required to the Urbane Cyclist. These changes greatly increased the success of the project. (Rob King)

# Portal

Interactive installation consisting of multiple video projections displaying time-based media (videos) whose behaviour (timing, order, volume, editing) can be controlled through motion. Motion is used as a 'natural' interface that allows users of multiple levels of computer/mobile literacy to access content. The user will be able to control the video behaviour with a modified Wii controller (first prototype) or SunSpot sensor (second prototype), camera-phone (third prototype) or other Bluetooth-enabled devices (future prototype).



## **Iterations:**

1. Play/Stop video: allow control of playback through manipulation of accelerometer sensor
2. Edit between videos: switch between simultaneously running video streams
3. Slow down playback: affect video stream to create slow-motion effect and/or reverse playback (enabling user to 'scrub' or search video for desired location)
4. Volume control: affect volume of one stream over another
5. Multi-screen control: affect streams on located on multiple screens
6. Future iterations:
  - Implement high-resolution accelerometer support (SunSpot sensor)
  - Implement camera-phone gyroscopic program to control video

## **Overall Architecture:**

1. User affects the playback of a wall of video projects (or other display) using either a Wii controller, a custom-built controller (SunSpot), or a camera-phone.
2. User with either Wii controller or custom-built controller will not require special software to control the installation.
3. User with camera-phone will need to download camera-gyroscope application to control the installation.

4. User's movements are translated into x, y, z coordinates of 3D space. These measurements are mapped onto corresponding control actions (play, stop, source A/B, volume, reverse, etc.)

5. Bluetooth server receives data broadcasted by controller and creates a patch to the video controller software on one or multiple computer/projector units.

### **Technical Development:**

Apr – Jun, 2007

Preliminary tests in the creation of a text analysis engine (custom built in Java) proved successful to drive the selection of video clips to be played. Preliminary tests in voice recognition using open source software (Sphinx) revealed that extensive work is required to 'teach' the engine and extend the library of recognized words. Difference in microphone models proves to greatly affect the accuracy of voice recognition. Latency issues in processing of speech and text analysis seem to be a roadblock to creating a responsive user experience. 'Natural' interface for interactive video installation is shifted to movement rather than voice in order to address latency, accuracy and platform issues discovered during testing. Initial tests are made with Wii controller and OSC (Open Sound Control) protocols via Bluetooth connection. Visual model of a Wii-embedded physical model is created as proof of concept for a 'natural' interface that works on a similar level as voice (non-digital user interface). Work is done on the receiving x, y, z (roll, pitch, yaw) measurements from Wii controller's accelerometer sensor. Accelerometer data is translated into playback controls: play/stop, edit between videos, playback speed and direction, volume control, multi-screen control is explored with multiple sensors. (Patricio Davila)

Built a proof of concept application for Portal, which parsed a text file containing a 100-word short story, and output a sequence of images that corresponded to the words in the story, thereby translating the story into visual form. To accomplish this, I created a Java

application which would break the story down into individual words, and built a database of images (using MySQL database). For each constituent word, the application would search the database for a matching image, and output it to a row on an HTML page if found. Therefore, after all words were processed, the HTML page would contain a sequence of adjacent images presented in the form of a filmstrip. (Ken Leung)

## Workshop in a Suitcase / Newspaper Box

A series of portable, easy-to-implement mobile development tools, including SMS servers, Wi-Fi modules, development laptops, phones and display technology. The Workshop was presented to designers in Peru and Canada and the assets were used to implement the Newspaper Box for public presentation in Toronto.

### Technical Development:

Apr – Jun, 2007

A small application has been written to transform a mobile phone into a Short Message Service server. All the elements required for this server are: a mid-end cell phone; a laptop; Bluetooth connection between both devices; cell phone plan including unlimited SMS messages.

- investigated code libraries for Java-based Processing development environment to access Bluetooth communications and SMS detection on cell phones

- Bluetooth

communications application created to send data from phone to laptop using Java Bluetooth library on phone and serial library on laptop



- SMS receiver written for phone and tested
- discovered that Java applications written for phone must have access to default SMS port to detect messages
- default SMS port is protected and not accessible by unsigned Java applications
- custom SMS port can be used to detect new messages but this port must be assigned in incoming message
- unable to assign custom SMS port in user phone using built-in SMS message composer
- SMS sender application written to assign custom SMS port to outgoing message
- phone to phone SMS is now working but only with custom applications on each phone
- investigated code libraries for Python for Mobile to explore possibilities of resolving SMS port issue and not requiring an application to reside on both server and user phone
- Bluetooth Python application and SMS receiver application have been hacked together and is now working (Patricio Davila)

Dec 2007

- Using existing prototypes that used Bluetooth to communicate between computer and mobile phone to establish the link to pass the contents of the text message to the application running on the computer and ultimately driving the larger interactive installation
- Repurposing an SMS receiver application in the Python scripting language on the S60 Nokia platform
- This Python application was merged with a Bluetooth transmitting application to create a seamless 'push' application on the phone, i.e. once any new SMS message was detected the mobile application sent the contents to the computer through its Bluetooth connection (Patricio Davila)

Feb 2008

“Newspaper” was an SMS/MMS driven instantiation of the techniques and software developed for Workshop in a Suitcase.

- Set up database for newspaper box applications in MySQL -- table (SOAPBOX\_MSGS) for MMS messages, and table SOAPBOX\_SMS for SMS messages
- Researched existing tabloid newspaper layouts to figure out a look & feel (blocky fonts, primary colors) that would work for our layout and designed rudimentary layout for newspaper box in Photoshop
- Tried using Python Image Library for the Newspaper Display module, but PyGame proved to be more useful/ versatile – different layout objects could be attached to movable layers
- Added MMS parser to display module, coded in Python, using email. Parser library to parse emails and extract images. Encountered some interesting MMS format issues – while MMS subject line is shown in the email by Telus, Rogers/ Fido MMS messages require clicking on a link to a separate webpage – therefore, decided to use text message as headline, instead of MMS subject message
- Added SMS parser to display module, which receives SMS messages from SMS phone via Bluetooth serial port
- Encountered some problems with COM port naming under Windows XP – discovered that COM ports higher than 8 must have a “..\..\COMx” format to be found
- Using MySQLDB library, added database functionality to display module – added code to SMS and MMS parsers to add new entries each time new content is received
- Tested display module with one central block (last MMS picture, with last SMS message overlaid)
- Due to bad visibility of text headlines on lighter images, added code that applied a black outline around all text
- Added text wrapping using render\_textrect library

- Tested power duration of chargeable battery pack with all components attached – yielded only 1.5 hours (2.5 hours of power total including the laptop’s built-in battery) (Ken Leung)

Mar 2008

- Added exceptions to prevent crashing due to: extra-long sentences, lack of internet connectivity, and bluetooth problems
- Separated MMS parser and SMS parser into individual applications – this way, they can be restarted independently and not crash the whole application if problems occurred
- Researched other wrapping libraries, since `render_textrect` was unable to handle very long words, wrapping only at whitespace
- Tried PyHyphen library (which uses a built-in dictionary to apply hyphenations at the proper spots in words) to perform wrapping, but it proved often ineffective for misspelled words
- Finally wrote wrapping code from scratch, which would always cut text off at a given line length – while not perfect (since some words would be cut off mid-syllable), it provided the most reliable and error-free solution
- Refined display layout to give more newspaper-ly feel
- Installed apps and environment onto second MacBook Pro (so that development machine wouldn’t be used in the field) (Ken Leung)