Multi-Perspective Real-Time Light Painting

Brett Holcomb
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
brett.holcomb@gmail.com

Mark D. Gross
COmputational DEsign Lab
Carnegie Mellon University
Pittsburgh, PA 15213 USA
mdgross@cmu.edu

ABSTRACT
In this paper I describe my project that uses multiple webcams, a laptop PC, an Arduino microcontroller, openFrameworks (an open source C++ toolkit), and multiple sensors to create light paintings in real-time that are viewable from multiple perspectives. I discuss the motivation behind the project and give some background information on similar work, go into detail about the implementation, and finally draw conclusions from the demo for directions for refinement and future work.

Author Keywords
Light-painting, interactivity, creativity, expressivity

ACM Classification Keywords
J.5 Arts and Humanities.

INTRODUCTION
For this project I wanted to create a light painting experience that allowed users to have feedback while painting, gave a sense of dimension by showing the light-painting from multiple angles, and also took advantage of tangible interaction to immerse the audience in the experience.

Related Works
This work is similar to Glowdoodle by Eric Rosenbaum at the MIT Media Lab [1], with the key differences being the multiple cameras and the physical interaction with my installation. You can go to glowdoodle.com and use your computer's webcam to make a light painting with constant feedback on the screen. My project uses the same simple algorithm as Glowdoodle: keeping the brightest pixel each time the camera captures a new frame. One of the best things about Eric's work is that anyone can go to the website and start creating immediately and then upload their painting to the online gallery.

My project was also inspired by "bullet time" [2]. This is an effect, popularized by the movie The Matrix, where multiple cameras are used to capture simultaneous images and then those images are played back sequentially to create the impression that time is frozen. For this project I wanted to give an impression of the 3-D shapes participants were creating by quickly showing them from multiple angles.

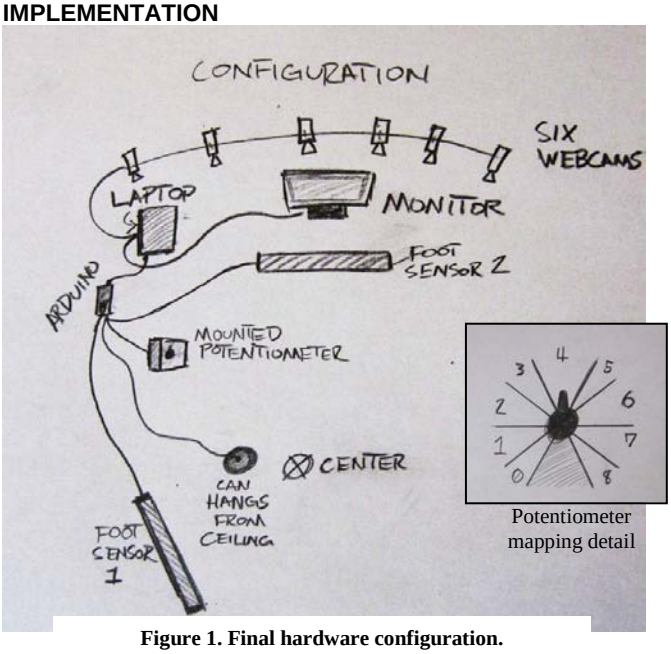
IMPLEMENTATION


Figure 1. Final hardware configuration.

In the following section I will describe the parts of the system in this order: a comprehensive list of materials and software, an overview of the sensors and the Arduino code, and finally an explanation of the openFrameworks code.

Materials List
- PC Laptop with Visual Studio C++ Express 2008 [4] and matching openFrameworks [5] toolkit installed, AMCap [6] also installed for debugging cameras
- Arduino Nano Ver 3.0
- 6 Logitech Quickcam Communicate 1.3MP webcams
- 2 step sensors (on the ground)
- 1 microphone in a coffee can (hanging from above)
- 1 potentiometer (mounted to a piece of foamcore)
- 3 10k ohm resistors
- Wire sufficient to connect sensors, approx. 20' total

Arduino Code

The Arduino accepted input from users using the 4 sensors and sent the following single ASCII characters to the computer via the USB serial connection: c, p, m, and the numbers 0-8.

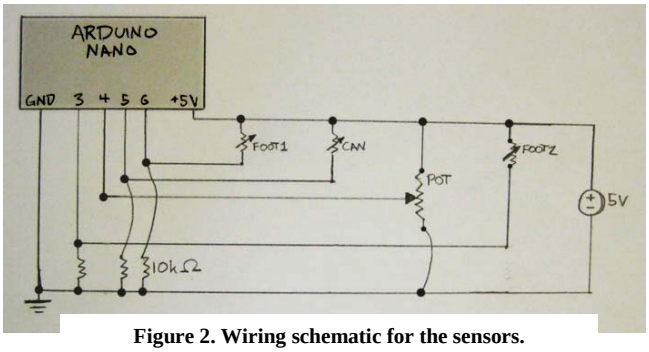


Figure 2. Wiring schematic for the sensors.

The character 'c' was sent when the user hit the coffee can hanging above them in the interaction space. A drumstick was provided and this was by far the most popular interaction with the system other than the actual light-painting. To implement this I mounted a simple inexpensive microphone to the inside of a coffee can and tested to see how the output changed when the can was hit. When the code detects a peak value it starts a timer to prevent duplicate signals from being sent. This is called debouncing. All additional peak values in the next 500ms are ignored.

The characters 'p' (pause) and 'm' (mode) were sent when foot sensors 1 and 2 were stepped on respectively. These sensors were also debounced. When a 'p' was sent to the computer all new images from the cameras were ignored and the user could take a screenshot by hitting the can. 'm' switched between the two rendering modes, either all 6 images shown in a grid or a fullscreen view of each image shown sequentially.

When showing the images sequentially, the current value from the potentiometer determined how quickly the images changed: from 0 (stopped) to 8 (very fast). I tested the potentiometer to create the mapping shown in Figure 1 with hard coded values because the values on the Arduino analog input pin didn't map linearly to the rotation of the potentiometer's knob.

When the images were shown quickly from each camera sequentially the participant got a good idea of the 3-D shape they had created. One of the biggest challenges I faced in the demo was encouraging some participants to move freely in the space provided. The effect was most impressive when they made large movements towards and away from the camera array.

openFrameworks Code

openFrameworks (OF), despite being a bit of a challenge to set up initially, provided a very powerful and usable environment in which to process the data from six webcams simultaneously. Each frame, it grabbed the data from the cameras and kept any pixels that were brighter than those in the existing images. Interestingly, performance was better in a dark room, possibly because the cameras were capturing fewer frames per second.

OF communicated with the Arduino to listen for the incoming characters and then provide all the functionality detailed in the previous section. For the numbers 0-8 the delay between images varied from infinite to 50ms. I also programmed a number of debug commands into the OF program to align the 6 cameras and bring up the hardware settings for each camera.

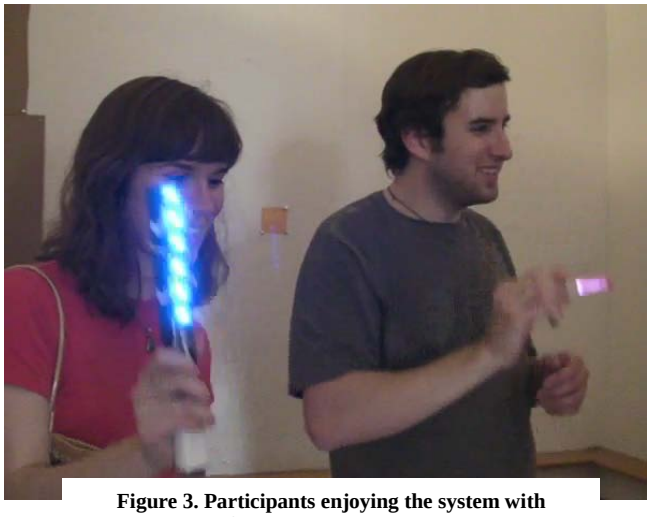


Figure 3. Participants enjoying the system with a couple of improvised light-paint brushes.

LESSONS FROM THE DEMO

During the demo there were a few key things that worked well and some others that stood out as having room for improvement.

Everyone really enjoyed hitting the can to clear the current image. This worked out well because it was definitely the tangible input that got the most use. Users weren't nearly as eager to use the foot sensors or the potentiometer. In the case of the foot sensors I think it wasn't obvious what they were for and the potentiometer just wasn't as entertaining as painting with light or hitting a can with a stick. I would occasionally demonstrate to the participants that they could use it to stop one any one of the six images, but no one seemed interested in turning the knob themselves.

A complaint that several people brought up was that they would rather see the images mirrored so that they were in the same orientation as the original painting. In testing, I was using light to write on the wall behind me, but this may

be the only circumstance in which a mirrored image is preferrable.

Overall, people seemed to engage fully with the interaction, but a few didn't move in the space very much initially and therefore didn't produce a very interesting sense of dimensionality with their images. Once I encouraged them to move around they generally seemed much more pleased with the results. The demo area also wasn't as dark as my test area so the light-paintings were generally a bit less vivid and awe-inspiring.

REFERENCES

1. Rosenbaum, E. Glowdoodle: A Medium for Expressive Inquiry. In *Proceeding of the seventh ACM conference on Creativity and cognition*, ACM Press (2009), 469-470.
2. Batten, C., et al. Toshiba "Time Sculpture". In *SIGGRAPH 2009 Computer Animation Festival*, ACM Press (2009).
3. Willis, K., User Authorship and Creativity within Interactivity. In *Proceedings of the 14th annual ACM international conference on Multimedia*, ACM Press (2006), 731-735.
4. Visual Studio C++ Express 2008. http://www.microsoft.com/express/
5. openFrameworks Visual Studio 2008 FAT vers. http://www.openframeworks.cc/versions/preRelease_v0.061/of_preRelease_v0061_vs2008_FAT.zip
6. AMCap 9.20, webcam testing software. http://amcap.en.softonic.com/
7. Mark The Dark. http://markthedark.com/