CPSC 490: Senior Project Report

Advisor: Professor Yang Richard Yang

Senior Project Report by Kenta Koga

“Falling Typography”

An Installation Interactive Art by Kenta Koga

Exhibited at an art show “32 Lilac Open House” 4/25 - 4/27, 2014
ABSTRACT:

This project uses the depth sensor that is equipt in Kinect, and recreate a three dimensional model of the world. It was an installation interactive art piece that projected a three dimensional point cloud of the exhibition space which included the viewer of the art, and also allowed the viewer to type in words or sentences via wireless keyboard connected by bluetooth. When the viewer typed in a word or a sentence, each letter that was typed would generate a sound, and fall to the bottom of the projection. Each letter is a physical object that follows the law of physics that is defined in the program. It falls, bounces off of another letter, spins, and settles on the ground. The algorithm also includes an edge detection of each frame so that the point cloud image that is generated would interact with the falling letters as well.

This piece of installation art aims to represent the inner emotion we carry when writing. Each letter piles up while the writer witnesses herself behind the letters of her choice. The piece used Processing as the programming language, with physics engine Fisica, geometrics engine Geomerative, Kinect data engine Open Kinect, and MIDI format sound generation engine Sound Cipher. As hardware, it uses Microsoft Kinect model 1414, and Apple’s bluetooth wireless keyboard. The piece was exhibited at “32 Lilac
Open House,” a local art show in New Haven curated by students in the Yale Art School from the April 25th to April 27th. In the basement of a house built by the Yale School of Architecture on 32 Lilac Street, New Haven, CT. It received extremely positive feedbacks, and it’s currently in the process of further improvement in order to be exhibited in couple of other exhibitions on campus towards the end of the semester.

The project was started in the beginning of January 2014, and completed on April 23rd, under the advisory of Professor Yang Richard Yang in the Yale College Department of Computer Science.

**STORY AND MOTIVATION:**

After studying various types of practical applications of computer science throughout my 3 years of study at Yale, my passion for my senior project was to work on a project related to creation of art. I’ve always been interested in artistic programming, but I never had the opportunity to study and actually be involved in creating an art piece that involved programming.

Above: Photo from a performance from a digital interactive performance group Enra

Inspired by exhibitions and performances by groups such as Enra (Japan) (http://enra.jp/) and Pietragalla Derouault Company (France), my initial idea was to create an interactive performing art
piece with a dancer. At the beginning of the semester, I explored possibilities from creating a visualization for dance by tracking the movement of a dancer, to creating an interactive visualization that dancers can interact with during the performance, but it was very difficult to coordinate with a dancer as they many of the highly skilled dancers who I knew already had schedule performances at the end of the semester. When I was thinking of the idea for pivoting, I ran into Rebecca Aston who was also a senior majoring in the fine arts.

Rebecca had an idea of hosting an art show at the end of the semester with some of her friends in the art department. The idea was that we would rent a house built by the Building Project (http://yaleherald.com/covers/one-house-at-a-time/) of the Yale School of Architecture, and turn that entire house into an art exhibition space. It aimed to raise awareness of the Building Project and to create a better relationship between downtown New Haven and students at Yale. I immediately signed up for it to become the core member of curating this art show.

To briefly explain the Building Project of the Yale School of Architecture, it is a project by the second year MArch (Master of Architecture) students in which the entire class goes through the process of designing to building a residential architecture in the New Haven downtown area from scratch. It is officially a part of the program of the Yale School of Architecture, and every year, these Yale students become in charge from designing to construction of an entire building. Two years ago, they built a house on 32 Lilac Street, New Haven, where we were able rent for our art show, through the New Haven Neighborhood Housing Services.

The moment I saw the exhibition space, I knew I had to do something in the basement. It was a very dark space with white walls that was perfect for projection.

I was inspired by the Advanced Graphic Design class that I was enrolled in this semester, and was starting to think that it would be interesting to create something that is related to typography. Also I hadn’t changed my mind from initial idea of using Microsoft Kinect and creating a visualisation from the data received from the sensors. Combining the two ideas, I decided to create the “Falling Typography.”
TECHNOLOGY:

1. Setting up Kinect and its Library

It was surprisingly difficult to set up the Kinect API on an Apple computer. Through research, I found out that Apple computer basically only works with Kinect model 1414 with Processing 2.1.

The library for Kinect to be used with Processing can be installed from here:

https://github.com/shiffman/OpenKinect-for-Processing

And there is a whole community around Open Kinect for Processing:

http://openkinect.org/wiki/Main_Page

2. Getting Data from Kinect

As it was unexpectedly difficult to get all the information necessary in order to accurately gather data from Kinect, I will summarise the methods I used below.

Currently, the OpenKinect Library for Processing 2.1 makes the following data available:

- RGB image taken from the Kinect camera as a PImage format
- Grayscale image from the Kinect IR camera as a PImage format
- Grayscale image each pixel’s brightness mapped to depth
- Raw depth data in 11bit number format in an int array

We can turn on/off each sensor with the following code:

enableRGB(boolean) : turn on or off the RGB camera image
enableIR(boolean) : turn on or off the IR camera image
enableDepth(boolean) : turn on or off the depth tracking
processDepthImage(boolean) : turn on or off the depth image processing
PImage getVideoImage() : grab the RGB or IR video image
PImage getDepthImage() : grab the grayscale depth map image
int[] getRawDepth() : grab the raw depth data
Example: getting the RGB Depth data from Kinect

3. Creating the Point Cloud

Once I got access to the raw depth data, I worked on creating the three dimensional model of the physical space using the depth data. With this, the algorithm would be able to recreate the model of the space in a complete darkness, as it only require depth information from the IR sensor.

Given the depth data, I assumed that I can draw points on the 3D space with Processing’s three dimensional capabilities. I defined the three dimensional space, and pixel by pixel I decide the (x, y, z) coordinate on which I draw the point. The most challenging part was translating the raw depth data obtained from Kinect to a physical depth value unit that we can understand.

I started off by simply measuring different length and obtained the raw depth values from Kinect, and by tracing back I tried to calculate the actual formula that translates the Kinect raw depth value into meters. However, probably due to the depth resolution, I wasn’t able to accurately come with the formula. In the end, through my research on the Internet, I came across the work by Matthew Fisher that provided the equation to translate Kinect raw depth value into meters.

(reference: http://graphics.stanford.edu/~mdfisher/Kinect.html)

depthInMeters = 1.0 / (rawDepth * -0.0030711016 + 3.3309495161);
With the above equation, I was able to draw points on the Processing’s 3D space with meter units. Also, as the resolution of Kinect depth sensor is 640px by 480px, it’s possible to visualise the point cloud in point cloud by scaling the original image differently.

Scaling down the Kinect resolution makes the point cloud more dense.

On the contrary, scaling up the Kinect resolution makes the point cloud more scarce.
4. The Fisica Physics Engine

In order to make each letter physically interact with each other, I used Fisica physics engine and Geomerative Library for Processing. Fisica is simply a Processing wrapper for JBox2D which is a commonly used physics engine for Java, and Geomerative library extends 2D geometry operations to facilitate generative geometry. Geomerative makes it easy to access the paths, the handles, and the points of a TrueType font, and makes it easy to develop generative typography and geometry pieces in Processing.

Fisica: http://www.ricardmarxer.com/fisica/
Geomerative: http://www.ricardmarxer.com/geomerative/

When each letter is typed, the algorithm converts the TrueType format font into a RShape format, creates the paths and edges, and throws the converted object (the letter that is turned into a Fisica object) into the physical world defined in the program.

The algorithm also allows the user to switch the font of the letters by pressing the Enter key during typing. All the fonts are stored in the data folder, and Enter key simply changes the font name to be called from the data folder.

The speed with which the letters fall, and the bounciness of the letters can be adjusted by modifying the setGravity() function in the Fisica set up. By adding other functions from Fisica in the set up, I can also make other physical reactions happen in the program.
5. The Sound Cipher Library

SoundCipher is a library that provides an easy way to create music in the Processing environment. With the SoundCipher library added to Processing, I was able to write a program that plays a sound along with my graphics.

More technically speaking, this library provides a simple interface for playing 'notes' on the JavaSound synthesizer, playing audio files, and communicating via MIDI. It provides accurate scheduling with the rest of the processing code, and allows events to be organised in musical time by using concepts such as beats and tempo. It also uses a 'score' metaphor that allows the construction of simple or complex musical arrangements. SoundCipher is designed to facilitate the basics of algorithmic music and interactive sound design as well as providing a platform for computational music.


In my program, every keyPressed() detection makes a call to playNote() in the SoundCipher library. It can either play a random note or a composed note depending on the setting of playNote(). Furthermore, by adding sound files to the data folder, it’s very easy to play those custom sound instead of a note.

6. Simple Edge Detection Algorithm

In order to have the falling letters bounce off not only from other letters but also from the point cloud, I attempted to run an edge detection on each frame and recreate the point cloud as a two dimensional image with transparent Fisica objects that maps the points. Although this wasn’t possible with the computing power of a MacBookAir, I will summarize the simple edge detection algorithm I used in my program. I used the simplified version of the Canny detection method, in which I analyze each pixel one by one in relation to all its neighbor pixels and bring up the contrast between the pixel and its neighbor pixels. By keeping track of the difference in color (or black/whiteness), this enables to sharpen the image and re-define the edges in the new image to be generated.
7. Result

By combining all the methods mentioned above, I was able to create and successfully exhibit the Falling Typography at an art show over the course of three days. The program was robust enough to keep running for the entire time when the piece was displayed, and I received extremely positive reviews from the visitors to the art show.

I would like to keep making improvements to the piece to make it into a better installation in the future. Some improvements that I intend to make in the near future include:

1. Enabling the interaction between the point cloud and the letters by using a more sophisticated edge detection method.

2. Somehow increasing the frame rate for Kinect so the animation will be displayed more smoothly.

3. Include more colors to both the point cloud and the letters in order to increase complexity of the piece.

It was a fantastic experience working on a artistic programming project for the first time, and it definitely inspired me to stay involved in this field and continue to study more. This project made me realise that there are more unexplored combinations of technology and art, and I’m determined to continue my exploration.

Special thanks to my project advisor
Prof. Yang Richard Yang
and my senior advisor
Prof. Dana Angluin

Yale College Department of Computer Science,
Senior Project by Kenta Koga, CS490, Spring Semester 2014
REFERENCES / BIBLIOGRAPHY:

Processing -
http://www.processing.org/

Setting Up Kinect -
http://blog.nelga.com/setup-microsoft-kinect-on-mac-os-x-10-8-mountain-lion/
http://synapsekinect.tumblr.com/post/6610177302/synapse
http://openkinect.org/wiki/Main_Page
http://social.msdn.microsoft.com/Forums/sqlserver/en-US/a1a91917-3e6a-4a8c-9308-029c40ffb704/kinect-sdk-for-mac
http://cargocollective.com/benMcChesney/Hacking-the-Kinect-GAFFTA
http://shiffman.net/p5/kinect/
https://code.google.com/p/simple-openni/
http://graphics.stanford.edu/~mdfishe/Kinect.html

Physics Engine -
http://bulletphysics.org/wordpress/
http://box2d.org/
http://www.jbox2d.org/
http://www.ricardmarxer.com/fisica/

Libraries -
http://explodingart.com/soundcipher/
http://www.ricardmarxer.com/geomerative/
http://www.ricardmarxer.com/fisica/

Edge Detection -
https://www.processing.org/examples/edgedetection.html
http://en.wikipedia.org/wiki/Canny_edge_detector
http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html