CSCI 4611: Programming Interactive Graphics and Games

Assignment 1: Text Rain

Handed out: Monday, January 30

Due: ~~Monday, February 13~~ Friday, February 17

Introduction

Interactive computer graphics can be used for many purposes. One of the most engaging is interactive art. This is one of the primary uses of the Processing toolkit that we have been exploring in class.

In this assignment, you will be writing an application in Processing that re-implements a neat interactive art installation, *Text Rain* by Romy Achituv and Camille Utterback (http://camilleutterback.com/projects/text-rain/). This is an engaging video-based digital installation that is in five permanent collections and has been exhibited dozens of times around the world. The artists themselves describe it best:

> *Text Rain is an interactive installation in which participants use the familiar instrument of their bodies, to do what seems magical—to lift and play with falling letters that do not really exist. In the Text Rain installation participants stand or move in front of a large projection screen. On the screen they see a mirrored video projection of themselves in black and white, combined with a color animation of falling letters. Like rain or snow, the letters appear to land on participants' heads and arms. The letters respond to the participants' motions and can be caught, lifted, and then let fall again. The falling text will 'land' on anything darker than a certain threshold, and 'fall' whenever that obstacle is removed.*

In this assignment you should learn to:

- Use the Processing environment for some serious programming.
- Work with 2D graphics coordinate systems and primitives.
- Handle game-like full-body gestural user input and respond via updates to the graphics displayed.
- Animate computer graphics based on data and user input.

Requirements and Grading Rubric

Before getting started, watch some of the videos of people interacting with Achituv and Utterback's *Text Rain* on the website mentioned above. Your job is to write your own Processing sketch to create a similar interactive experience, mimicking the interface of *Text Rain* as much as possible.

To get you started and help with the logistics of getting video data into Processing, we provide some support code (see section 5 for details) that will read live video from a webcam attached to your computer *or* use a pre-recorded video file as input. You need to pick text characters from some text of your choosing using a random or pseudo-random algorithm and then make them fall from the sky like rain. You'll have to set their initial *x* and *y* coordinates on the screen, draw each character using Processing's text drawing routines, and then advance the *x*, *y* positions over time to create an animation. The text characters should fall like rain until they bump into a dark object at which point they should stop falling. As in the original art installation, we will assume that people show up in the video as dark objects (the color values for these pixels will be close to black) and the background shows up as a light object (the color values will be close to white).

A more specific list of requirements follows. We use this list in our grading, so this also serves as a grading rubric. To get a 100% on the assignment, you need to correctly implement everything that is listed here. To get a grade in the "A" range, you need to implement everything in the "C" and "B" ranges, plus some portion of the features in the "A" range. And so on. You'll find that the requirements are ordered very conveniently here… almost like a set of steps that you should take to complete the assignment. We recommend starting at the top and going down the list as you work.

Work in the "C" Range Will:

- Draw video frames to the screen (basically already done by the support code).
- Flip the video image displayed on the screen so when viewers look at the display it looks like they are looking in a mirror at themselves.
- Load a font into Processing.
- Use the font to draw characters on the screen so they show up on top of the video.
- Animate the characters, making them fall down the screen.

Work in the "B" Range Will:

- Support debugging and setting a good threshold by displaying the results of a threshold computation – when the spacebar is pressed, toggle between displaying the final user view and a "debugging" view that shows a binary image for the foreground/background classification: all pixels classified as foreground should be pure black, all pixels classified as background should be pure white.

- Support interactive threshold adjustment for situations with different lighting conditions by changing the value of the threshold used to determine foreground and background objects interactively using the up and down arrow keys.
- As in Achituv and Utterback's implementation, display the video image in grayscale and the text rain in color.
- Make the characters stop falling when they "land" on a dark object whose brightness is below the threshold.

Work in the "A" Range Will:

- Include some variation in the velocity of the raindrops to make it look a bit more interesting.
- Use the real (wall clock) time to make the rain fall based on a velocity specified in pixels per second rather than just making it fall based on how many times your draw() routine is called per second.
- Make the rain falling algorithm "robust" so that rain doesn't just come to a rest when it lands in the palm of a viewer's outstretched hand but will also rise up with the hand if the viewer raises his/her hand.
- Make it so that the rain can "flow" down a slope by checking not just the pixel under the character but also some pixels to either side of it.

Additional Technical Background and Tips

The problem of identifying foreground and background objects in a video stream is very difficult if you try to solve it in a general sense. We are making it much easier in this assignment by making the assumption that the background is white and the people in the foreground will show up as dark objects. If you try this with your own webcam, then the best way to do this is to stand in front of a white wall. For this assignment, we will assume that we are always working with "good" input data that meets this specification – the background will be a white wall, and the foreground objects will show up as darker objects in front of the wall. This way, all you have to do to separate foreground objects from background objects is determine a good threshold value. For colors from 0 to 255, you might start by picking 128 for your threshold. Then consider any pixel that has a brightness less than 128 to be foreground, and any pixel with brightness greater than 128 to be the white wall in the background.

With this assumption in place, the remaining difficulty that you may encounter with input from a video stream is that video data can be noisy. In other words, even if the person in your video doesn't move at all, the pixel color data returned by your camera will likely fluctuate a bit. To reduce the impact of this in your calculations, you might find it useful to apply Processing's built in `blur` filter. If you smooth out the image by blurring it a bit before using a threshold to determine foreground and background objects, then this will reduce the impact of noisy input data.

Above and beyond

All the assignments in the course will include great opportunities for students to go beyond the requirements of the assignment and do cool extra work. We don't offer any extra credit for this work – if you're going beyond the assignment, then chances are you are already kicking butt in the class. However, we do offer a chance to show off… While grading the assignments, the TAs will identify the best few examples of people doing cool stuff with computer graphics. After each assignment, the students selected will get a chance to demonstrate their programs to the class!

For this assignment, you could for example consider adding collision detection between the characters, by checking that you do not move a character to a location where another character already is. This way, the rain will pile up rather than collapse into a single layer of characters (although it may behave more like a sand pile than a puddle).

Support Code

The webpage where you downloaded this assignment description also has a download link for support code to help you get started. The support code for this assignment is a simple Processing sketch which you can continue to add to as you develop your solution. To run it, you may have to install the Video library, which you can do by going to *Sketch* > *Import Library…* > *Add Library…*, selecting "Video | GStreamer-based video library for Processing", and clicking *Install*.

The support code displays a menu on the screen that you can use to select the input to use for your sketch. The default choice is menu item '0', which you can activate by pressing zero on your keyboard. This will use the file "TextRainInput.mov" located in the sketch's data directory as input. Alternatively, if you have a webcam attached to your computer, Processing will detect this and you will see up to 9 additional menu items numbered 1 to 9, one for each of the web cam devices identified on your computer. Press the number of the device you want to use.

We setup the support code this way for a couple of important reasons. First, the pre-recorded video is very useful for testing, since it provides a repeatable input stream. Second, we want you to be able to complete this assignment even if you don't have a webcam attached to your computer or a nice white wall to have your friends dance in front of :). Finally, we are going to use this pre-recorded video feature to grade your work in a consistent way. We will input our own video in the same style as the examples we distribute with the support code when we grade your project.

Handing It In

When you submit your assignment, you should include a README file. This file should contain, at a minimum, your name and descriptions of design decisions you made while working on this project. For example, if you developed your own algorithm for intelligent pseudo-random selection of characters from the text to make words show up in the rain then make sure you describe this algorithm in your README file. If you attempted any extra “above and beyond” work, you should note that in this file and explain what you attempted.

When you have all your materials together, zip up the source files and README, and upload the zip file to the assignment hand-in link on our Moodle site, which you will see on our calendar page for the day that the assignment is due. You do not need to include the data directory, which is fairly large and which we already have a copy of. Any late work should be handed in the same way, and points will be docked as described in our syllabus.