CSCI 4611: Programming Interactive Computer Graphics and Games
Assignment 1: Text Rain

Handed out: Tue 9/8
Worksheet Due: Sun 9/13 at 11:59pm
Program Due: Sun 9/20 at 11:59pm

1. 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 one of our favorite interactive art installations, *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 three permanent collections and has been exhibited dozens of times around the world. The artist describes 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.

2. Worksheet

Before jumping in to the programming portion of the assignment, we first ask that you complete a conceptual worksheet. This worksheet will cover some of the computer graphics topics that are used in this assignment, which will make it easier when you're writing the actual code! Look at section 7 for more details about how to get the worksheet and how to submit it.

3. Requirements and Grading Rubric

Before getting started, watch some of the videos of people interacting with 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 6 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).
- 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:

- Make the characters stop falling when they "land" on a dark object.
- 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.
- Support debugging and setting a good threshold by displaying the results of the threshold computation – when 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.
- As in Utterback's implementation, display the video image in grayscale and the text rain in color.

- 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. This must apply to *both* live video and pre-recorded video files.
- Pick characters to display (i.e., new rain to add to the scene) at random from some text that is artistically appropriate for the *Text Rain* theme.

Work in the "A" Range Will:

- Make the rain falling algorithm "robust" so that rain doesn't pass through black regions even if they are very thin (e.g., just a pixel wide).
- 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.
- Use a useful, more sophisticated algorithm (you should come up with your own idea here and ***describe it in your README file***) for picking characters to display at random so that the rain still looks random but there is a reasonable likelihood of catching (or seeing) a set of raindrops that spell out a whole word.
- Include some variation in the velocity of the raindrops to make it look a bit more interesting.

4. 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 or equal to 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.

There is no programming magic that goes into creating the "video mirror" effect – a simple iterative procedure is fast enough and relatively elegant.

4.1 Additional instructions for Linux users

If you run Processing on Linux, you may encounter this error:

A library relies on native code that's not available. Or only works properly when the sketch is run as a 32-bit application.

To fix it, you will need to:

1. Download **video.zip** from: https://github.com/gohai/processing-video/releases/download/v1.0.2/video.zip
2. Replace the contents of **home/sketchbook/libraries/video** with the contents of the newly extracted **video.zip** folder

5. Wizards

All of 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 4 or 5 examples of people doing cool stuff with computer graphics. We call these students our "wizards", and after each assignment, the students selected as wizards will get a chance to demonstrate their programs to the class!

For this assignment, you could for example consider making it possible for the rain to "flow" down a slope by checking not just the pixel under the character but also some pixels to either side of it. For example, if the pixel under the character is dark but the one to the left of it is light, the character could move to the side instead of going up. Or, you might 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). Or, even better, think of your own cool, creative idea!

6. Support Code

Support code can be obtained from our course GitHub organization. Check out section 7 for details on this. To run the code, 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.

7. Setting up GitHub and Handing In

For this course, all support code and worksheets will be provided using our course GitHub organization. Follow these steps to get your repository set up on you development machine:

1. Clone your repository for this class. *Put it inside a directory path on your computer that does not include any spaces!* On Windows, we recommend using Git-bash rather than the built-in Windows Command Prompt; Git-bash gets installed on Windows when you install Git. On OSX, you can use Terminal. On Linux, use your favorite shell. Then, type the following command:

```
git clone https://github.umn.edu/umn-csci-4611-s20/repo-[x500].git
```

2. Add remote server for our support-code:

```
cd repo-[x500]

git remote add upstream
https://github.umn.edu/umn-csci-4611-s20/shared-upstream.git
```

3. Pull from shared-upstream into your branch:

```
git pull upstream support-code --allow-unrelated-histories
```

4. Git add everything that was just merged from the upstream support-code branch into your own master branch, commit these new changes, and then push the result back to your own origin.

```
git add -A
git commit -m "merged in the latest support code"
git push origin master
```

You should now have all the support code in your repository (including directories like "dev" and "worksheets"). Take a look in the "worksheets" directory – you should see "a1_textrain.md" – **this is the file you will edit to turn in the worksheet**. You must directly edit this markdown file in order to receive full points on the worksheet. The Processing starter code is located in "dev/a1_textrain."

As you make progress on your worksheets and programs, make commits to your repository and push them. If you're unfamiliar with git version control, check out this excellent guide from GitHub.

When you submit your assignment, you should include a README file in the "dev/a1_textrain" folder. 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 "Wizard" work, you should note that in this file and explain what you attempted.