CISC 140: Lab 1

Contents

1 Introduction

This programming lab will test you on the basic programming concepts from CISC 120, and the review unit of this course. Additionally, it will focus on working with the `Picture` class and `stddraw` module used throughout the textbook. You should have a reasonable understanding of the basics of using (but not writing) classes and objects, modular programming and functions, working with arrays, and basic program control.

The project for this lab is based upon several of the Creative Exercises from Section 3.1 in the textbook. The general problem statement is as follows:

> Create a program which accepts a sequence of image filenames as command-line arguments, and displays these images one after another, showing each image for 2 seconds before moving on to the next. The slide-show should smoothly transition from image to image by fading from one image to the next.
>
> Then, modify this slideshow by applying a filter to each image. You may select one filter from the following: rotational filter, swirl filter, wave filter, glass filter.
>
> Be mindful of applying good, modular design techniques when composing this program, taking full advantage of functions.

2 Creating a Slideshow

As a starting point, we will create a program that displays a series of images, received as filenames from command-line input, one after another–holding each image in view for 2 seconds. Then we can move on to the more complicated application of effects to the images.

Displaying a Single Image

Recall that you can use the following code fragment to display a single image for one second.

```
# let filename refer to the file name of an image
pic = Picture(filename)
stddraw.setCanvasSize(pic.width(), pic.height())
stddraw.picture(pic)
stddraw.show(1000)
```

As a first step, take this code and generalize it so that it will display any valid image file, passed in as the first command-line argument to the program, for two seconds. **5pts.**

Displaying a Sequence of Images

Once this has been done, we'll need to think about how we can display an arbitrary number of images. Recall that the `sys.argv` object that we use to get the arguments is an *array*, and thus you can use a counting loop to iterate over it and perform some operation on each element within it.

```
for i in range(len(sys.argv)):
    process(sys.argv[i])
```

The first element is the program name, which isn't useful–but every element thereafter should be a valid image file.[1] So, as a next step, modify the program you wrote above to use a counting loop to iterate over several possible input images, and display each one for two seconds in turn. When designing your loop, make sure to account for the fact that the element at `sys.argv[0]` will not be a valid image.

CAUTION 1 *Be careful with what code you place inside of the loop, and what code you place outside. Because of the way that stddraw is set up, you cannot resize the canvas by repeatedly calling* ***`setCanvasSize`****. This is something you can only do once.*

You should be able to call the program like so

```
% python slideshow.py picture1.png picture2.png ... picturen.png
```

and see a slideshow of all of the specified pictures. **5pts.**

[1] If not–that is the user's problem, not yours.

Testing the Slideshow

On Moodle, you will find an archive containing several test pictures for you to use for this lab. First, test the program using the pictures titled `picture_1.png`, `picture_2.png`, and `picture_3.png`. The program should work as expected.

Next, try it out throwing `picture_4.png` into the mix. You should see an error. Describe in the space below what this error means.

No error when i ran the program only issue is that picture 4 does not show the whole image. this is due to the this is due to the canvas problem and the image being too large. So i have to apply some-sort of scaling to the problem for the issue to be resolved.

Fix this error in your code. Your slideshow should work for any and all combinations of the provided image files. You may want to check out page 376 of the textbook, if you have absolutely no idea what to do here. It will involve manipulating the image in some way prior to displaying it. **5pts.**

Applying the Fade Effect

At this point, you should have a working slideshow. However, we want our slideshow to also apply a fading effect between images, rather than having an abrupt transition.

This will require adapting some code from the textbook. Check out Program 3.1.7 on page 377. For this step, convert that program into a function called `fade`, which accepts as input arguments the currently displayed picture, and the picture that you would like to display next. This function should then apply a gradual fade effect (the specifics of how many steps, how long, etc., are up to you).

You should be able to call this function like so,

```
fade(current_pic, next_pic)
```

and place this function call within your loop somewhere. **15pts.**

3 Applying a Filter

Now that you have the slideshow fully working, let's mix it up a bit by adding a simple filter to the images. You will need to select one filter from the following list, and apply it to all images in your slideshow. The filter should be applied prior to the fade–so that the fade-in process uses the filtered image, and not the original one.

1. rotational filter
2. wave filter
3. glass filter
4. swirl filter

You can find descriptions for each of these filters within the Creative Exercises for Section 3.1 in the textbook.

Implement the Filter

Implement your selected filter as a function, which accepts the image to be filtered as input. The filter can either operate on the input argument in-place, or leave it untouched and return a new Picture object. Either way, call this filter function within your program's loop, prior to fading into the picture. **20pts.**

4 Analysis

This program has a few different "dials and knobs" that we can turn to tweak its behavior. We'll focus mostly on the slideshow itself here–though I would encourage you to also experiment with the various parameters that govern the way that your selected filter works. You can have a lot of fun playing with these!

Fading Parameters

Your `fade` function, developed earlier to handle fading images in and out, has two main parameters that govern the behavior of it: n and t. n describes the number of discrete "steps" the fade occurs in, and t is the amount of time that the picture for a given step is displayed.

(a) If you wanted the fade to be as smooth and seamless as possible, what sorts of values should you use for these two parameters? **5pts.**

so since n is the number of steps and t is the amount of time each picture is shown. to make a smooth transaction the n should be 2 steps and t should be 2. to get the best fade between images. you do not want the image to take to long fading or to little time between images

(b) Try adjusting your program to use these parameter values. Make the fade as smooth as you can. Do you notice anything odd about how the fade appears and how long it takes, compared to your specified values for n and t? If so, what do you think is causing this odd behavior? **5pts.**

I had 2 for n and 2 for t when testing this out i found that 1 for n makes the program look like there is no fading at all it just jumps to the next image. so i have it 1 more ,for the t I found that 2 makes it makes the smoothest fade effect. when i first started i did see something odd i had the n =1 and it would show no fade i also had 1 for t and the image would stay for 1 second so it would just run through the prorgam and not really work as intended. The reason this was accuring was because there was not enough t and n was not enough.

Code Profiling

One common task faced by programmers is the determination of how long it takes for a program to run, for a given set of inputs. Specifically, the programmer may be interested in *what part of the program is the slowest*. If you want a program to run quickly, identifying the slowest portion of the code–the bottleneck, as it is commonly called–is an important first step. It doesn't make much sense to spend hours working on speeding up part of the code that runs in 10 milliseconds when the part of the code directly before it runs in 4 seconds. This is the root idea of the famous Knuth quote, "Premature optimization is the root of all evil."

There are a variety of tools and techniques for doing this. Shortly, we will formally discuss a few of them, including empirical testing and mathematical analysis. The latter is largely preferable for reasons that we'll get into later, but the former is substantially easier to dig into. So we will do that here.

The idea is that we will time our code, and certain portions of our code, to determine how long it takes them to run for a given set of inputs. This will allow us to determine where the bottlenecks in our program are, and may help us to identify the source of any odd behavior you may have noticed in the previous question.

To do this, we will use Python's `time` module. This module contains a function called `time.time`, which returns the current system time in seconds, as a float. If we call this function, then have Python execute some code, and then call the function again, we can get a (very rough) idea of how long it took for that code to run.[2] The following code snippet shows how this might be done.

```
import time
import stdio

start = time.time()
do_stuff()
stop = time.time()

stdio.writeln(stop - start)
```

(a) For a start, let's try profiling your code. Pick a static set of inputs and parameters–you want to keep everything as constant as possible–and then use the above technique to profile parts of your program. Table 1 contains the parts that you should profile, and gives you a spot to write the times down. You may want to time each portion several times, and then write the average value in the table. **10pts.**

> **CAUTION 2** *It is **very** important for this step that you leave everything exactly the same. Call the program with the same input arguments, leave the fade*

[2] This technique is pretty rudimentary, but works well for example purposes. If you want to get serious about timing your code, you should look into Python's timeit module, which provides a far more robust implementation of this same idea.

CAUTION 2 (Continued)
and filter parameters the same, etc. The only changes from run to run should be the location of the timing code.

Table 1: Parts of your program to profile

Code Portion	Time (s)
Whole Program	15s
Displaying a Picture	2s
Filtering a Picture	3s
Scaling a Picture	4s
Fading a Picture	0.105s
Loading Picture to Background Canvas	0.794s
Displaying Background Canvas Contents	3s

(b) Of all of the individual components that you have profiled, can you identify the bottleneck, if one exists? Don't consider the first two parts listed on the table–look only the individual operations listed. Write your answer, and why you think that this particular piece might be the bottleneck, below: **5pts.**

the biggest bottleneck is scaling a picture. I believe this is because the program has to load the image and resize the entire mage to the specified dimensions. you also filtering and display tied for second these are pretty high since the program has to filter an image take the original and add the filter to it displaying background canvas contents has to grab grab the image and load it on to the canvas

(c) Now that you've identified a potential bottleneck, we can examine how variations in input affect its runtime. First, list below all of the parameters that you think might influence the runtime of the bottleneck. Don't forget to consider the size of the canvas being displayed! **5pts.**

scaling a picture was the biggest bottleneck. Again this is due to an image having to be resized. I belive for my program this took longer since i specified that the canvas size be the size of the image to support the 4th image. so the program has to scale each image to its original size. stddraw.setCanvasSize(width, height) this was the parameter. This also tied to the size of the canvas being displayed

(d) Pick the size of the canvas, and one other factor that you listed above, and profile the bottleneck using a variety of values for these parameters. Pick one to vary and leave everything else constant, then record the time for each tested value of that parameter. Then, switch to the other parameter to be tested and do the same thing–leaving everything else constant.

Once you have these times collected, plot the runtime vs. each parameter in Excel, or some other plotting utility. Do you notice any regularity or pattern in the way that runtime changes with these parameters? If so, can you come up with a mathematical function that describes this relationship? Describe your results, and your thoughts about them, in the space below and on the next page. Don't forget to answer all of the questions asked in this paragraph, and **make sure to make plots!** No plots means no points for this entire question. **10pts.**

i will submit the the excel sheet i don't know how to insert an image to pdf.

again since the bottleneck and the canvas size we're tied together. since i made the canvas the size of the image.

I changed it for the canvas to be 800x800, and 600x600 I recorded the original time vs the 800x800 and 600x600. i then did average of the times. they had similar times, but the a set size of the 800x800 and 600x600 were bit faster due to the program not having to set the canvas to the picture size. The canvas was set with a size and the picture size would not matter. with the scaling it was similar since the program not having to look for the image size it would make the image the size. i will submit the excel sheet. since i was not able to add it in the pdf.

(e) If you wanted to improve the performance of this program, what part of the code would you focus your attention on? Additionally, are there any minor tweaks you can make to the parameters of the program that could improve performance without modifying any of the core algorithms? **10pts.**

To improve the performance of the program I would focus on the scaling and the canvas size. I made the canvas size be the size of the picture and this made the program run longer. Since it had to set the size of the canvas to the size of the image I would just have a set canvas size for all images. making the canvas larger than all the images so the program would not have to reset the canvas after every image. making these changes would not affect the core algorithms.

5 Rubric

Question	Points	Score
Displaying a Single Image	5	
Displaying a Sequence of Images	5	
Testing the Slideshow	5	
Applying the Fade Effect	15	
Implement the Filter	20	
Fading Parameters	10	
Code Profiling	40	
Total:	100	