CSC 225 - SUMMER 2015
ALGORITHMS AND DATA STRUCTURES I
PROGRAMMING ASSIGNMENT 4
UNIVERSITY OF VICTORIA

Due: Tuesday, July 14th, 2015 at 4:00pm.

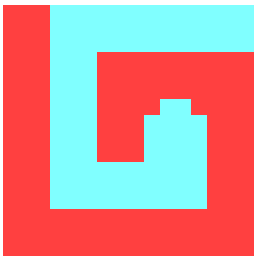
1 Images

Images can be represented by 2-dimensional arrays of pixels, each with a colour value. Usually, colour values are expressed as a triple (r, g, b) of red, green and blue values. Many image processing tasks work entirely with individual pixel values. For example, to convert an image from colour to greyscale, the colour value of each pixel can be converted to a shade of grey. For a pixel with colour (r, g, b), one possible conversion formula is to produce a new colour (r', g', b') where

$$(r', g', b') = \left(\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3}\right).$$

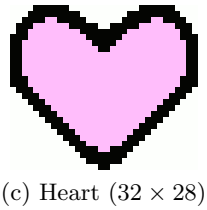
Many image processing tasks use information from neighbouring pixels, or from the entire image. CSC 225 will only cover one technique for image processing. The Department of Computer Science offers an entire course (CSC 205) on 2d graphics and image processing, which covers a broader range of techniques.

Consider the small images in Figure 1 below.



(a) Spiral (16×16)

(b) Rainbow (50×32)



(c) Heart (32×28)

(d) Flag of Canada (64×32)

Figure 1: Four small images.

Each image can be viewed as a grid of pixels. Within an image, groups of adjacent pixels with the same colour are often significant. This assignment, and assignment 5, will cover various algorithms to process contiguous regions inside images. Figures 2, 3 and 4 show the individual pixels of three of the images in Figure 1 above.

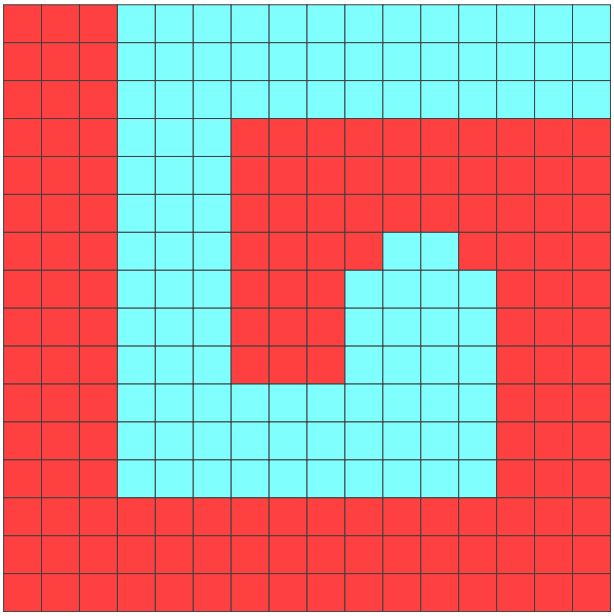


Figure 2: A close-up view of Figure 1a.

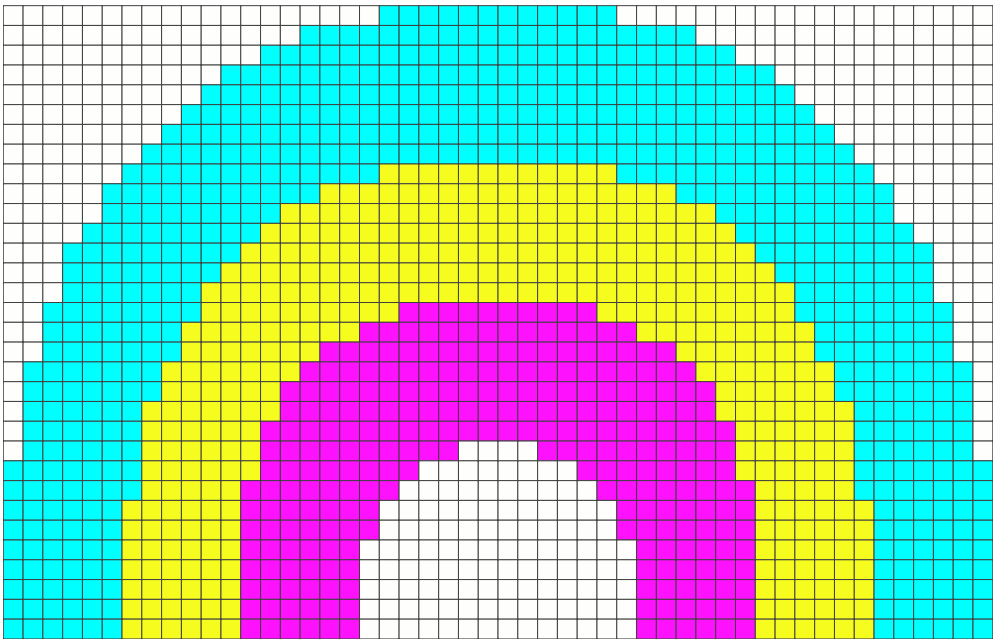


Figure 3: A close-up view of Figure 1b.

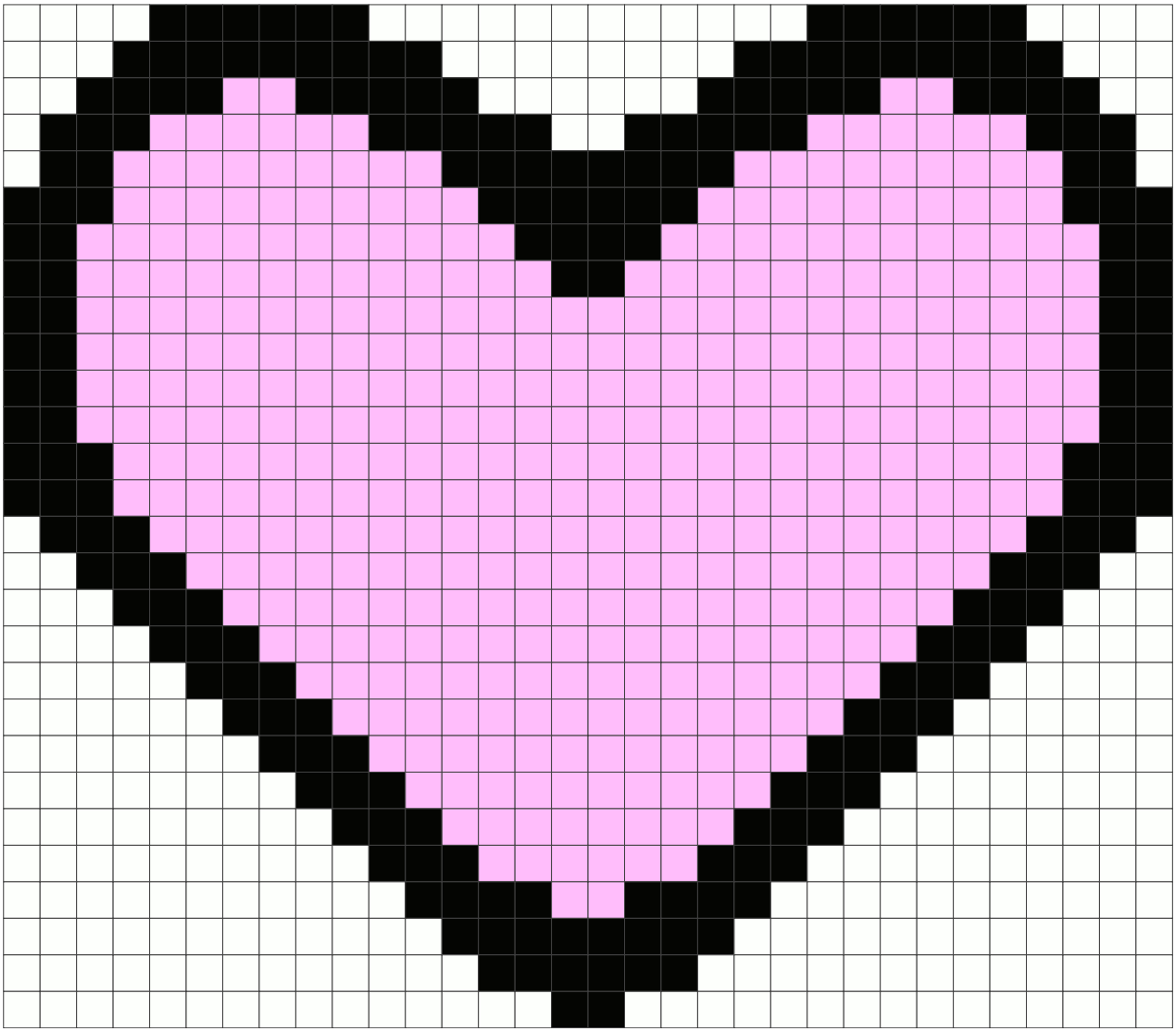


Figure 4: A close-up view of Figure 1c.

Define the *pixel graph* of an $n \times m$ pixel image to be an undirected graph with a vertex v_{xy} for each pixel (where $0 \leq x \leq n-1$ and $0 \leq y \leq m-1$), and edges between all pairs of neighbouring pixels with the same colour value. Figures 5, 6 and 7 show the pixel graphs for the images in Figures 2, 3 and 4.

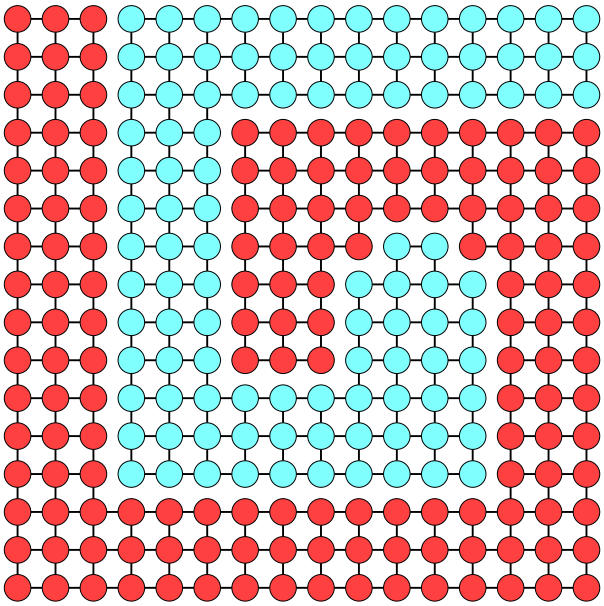


Figure 5: The pixel graph for the image in Figure 1a.

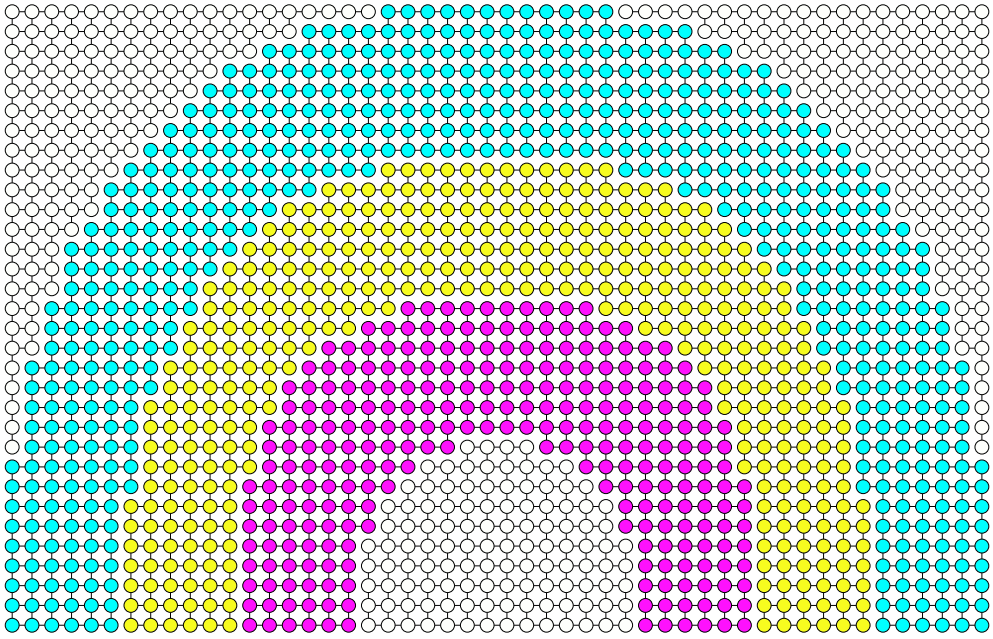


Figure 6: The pixel graph for the image in Figure 1b.

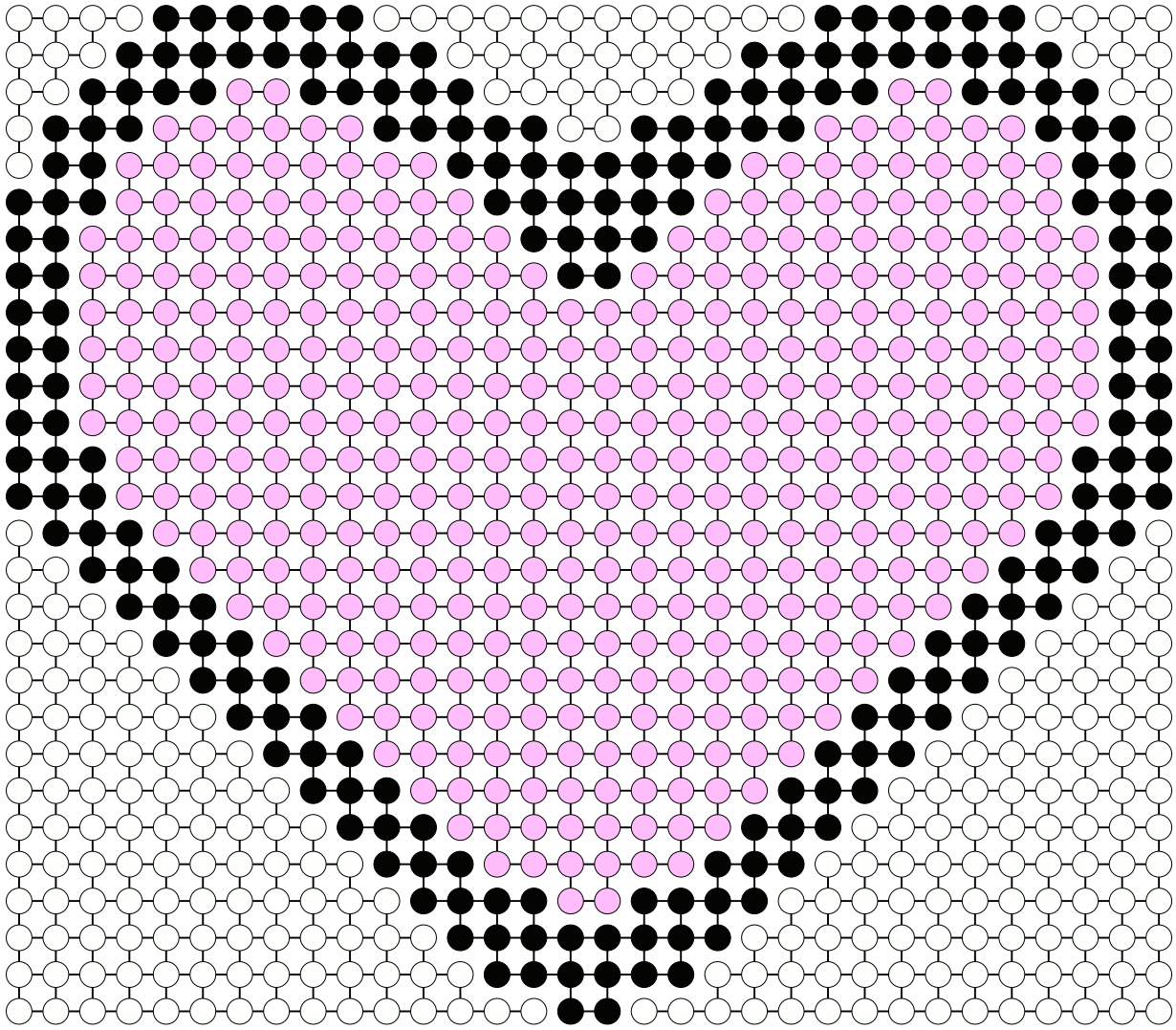


Figure 7: The pixel graph for the image in Figure 1c.

Each connected component of the pixel graph corresponds to a contiguous region containing a single colour in the original image. Vertices with degree 4 correspond to pixels in the interior of their region, while vertices with degree less than 4 correspond to pixels on a boundary.

2 Programming Assignment

The programming assignment is to implement a data structure to store pixel graphs, as well as an algorithm to construct a pixel graph from an input image (supplied as an array of (r, g, b) pixel values) and an algorithm to outline the regions of the image by changing the colour of pixels whose corresponding pixel graph vertices have degree less than 4.

The template for this assignment is divided among four files. You are required to use the provided

files as the basis for your submission, and may not change any of the classnames or method signatures inside any of the files, or your submission will not be marked. You are free to add additional classes, methods or instance variables as needed, except as indicated below. You are also permitted to include extra files in your submission. Note that if your submission is missing any files and does not compile, the error will be treated the same way as any other compile error (and you will not be given the opportunity to submit the missing files to recover the marks).

An image can be viewed as a 2d array of colour values corresponding to pixels. For this assignment, instances of the `Color` class in the `java.awt` package[1] will be used to represent colour values. An $n \times m$ image will therefore be represented by an $n \times m$ array of type `Color`.

`ImageViewerA4.java`	A graphical interface for viewing and transforming images. This is the 'main program' which uses the functionality defined in the other files. It is not necessary to read or understand the code in this file to complete the assignment (although you are encouraged to do so). You are free to modify this program for debugging purposes, but all of your modifications will be discarded before marking. During marking, the original, unmodified `ImageViewerA4.java` will be substituted. You will lose marks if your code does not function correctly (or if a compile error occurs) with the unmodified version of `ImageViewerA4.java`. **Methods to implement**: None.
`OutlineBoundary.java`	Contains a single static method `OutlineBoundary`, which takes a PixelGraph object (see below), a reference to an `ImageViewerA4` object (which is displaying the current image) and a particular colour C as parameters. Your implementation of `OutlineBoundary` must change the colour of all pixels corresponding to vertices of degree 3 or less to the colour C. Read the comments in `OutlineBoundary.java` for more details. **Methods to implement**: `OutlineBoundary`.
`PixelGraph.java`	The `PixelGraph` class implements a data structure for storing a pixel graph. **Methods to implement**: Constructor, `getPixelVertex`, `getWidth`, `getHeight`.
`PixelVertex.java`	The `PixelVertex` class implements a data structure to store each vertex of a pixel graph. **Methods to implement**: Constructor, `getX`, `getY`, `getNeighbours`, `addNeighbour`, `removeNeighbour`, `getDegree`, `isNeighbour`.

3 Image Viewer Interface

After compiling all four template files, the graphical interface can be started with the command

1. Documented at `http://docs.oracle.com/javase/8/docs/api/index.html?java/awt/Color.html`

```
$ java ImageViewerA4 image_name.png
```

Figure 8 shows the image viewer window displaying the file `rainbow.png` (shown in Figure 1b).

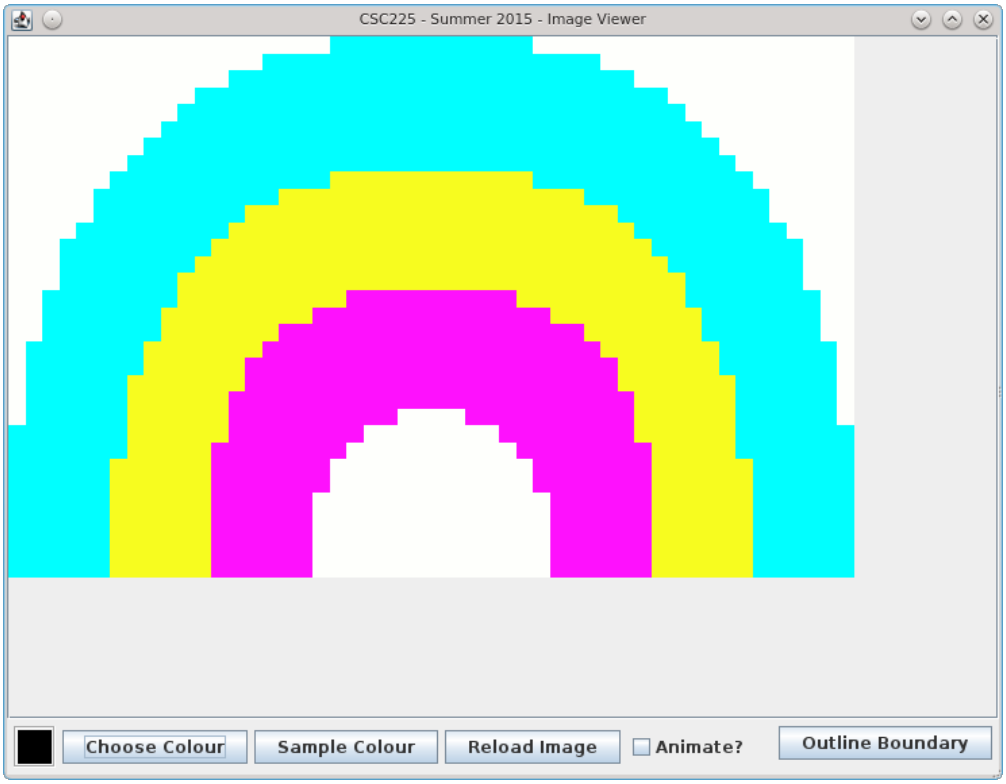


Figure 8: The image viewer window after opening the `rainbow.png` image.

The mouse wheel can be used to control the zoom level of the displayed image (since most of the test images are small, you will likely want to zoom in on them). The "Outline Boundaries" button constructs a PixelGraph object for the displayed image and passes it to the `OutlineBoundaries` function. The colour used for the outline is displayed at the lower left of the viewer window. To select a different outline colour, use the "Choose Colour" button. The "Sample Colour" button will allow you to set the outline colour from a pixel in the active image (by left-clicking on the pixel). Figure 9 shows the result of a model solution after clicking the "Outline Boundaries" button on the image from Figure 8.

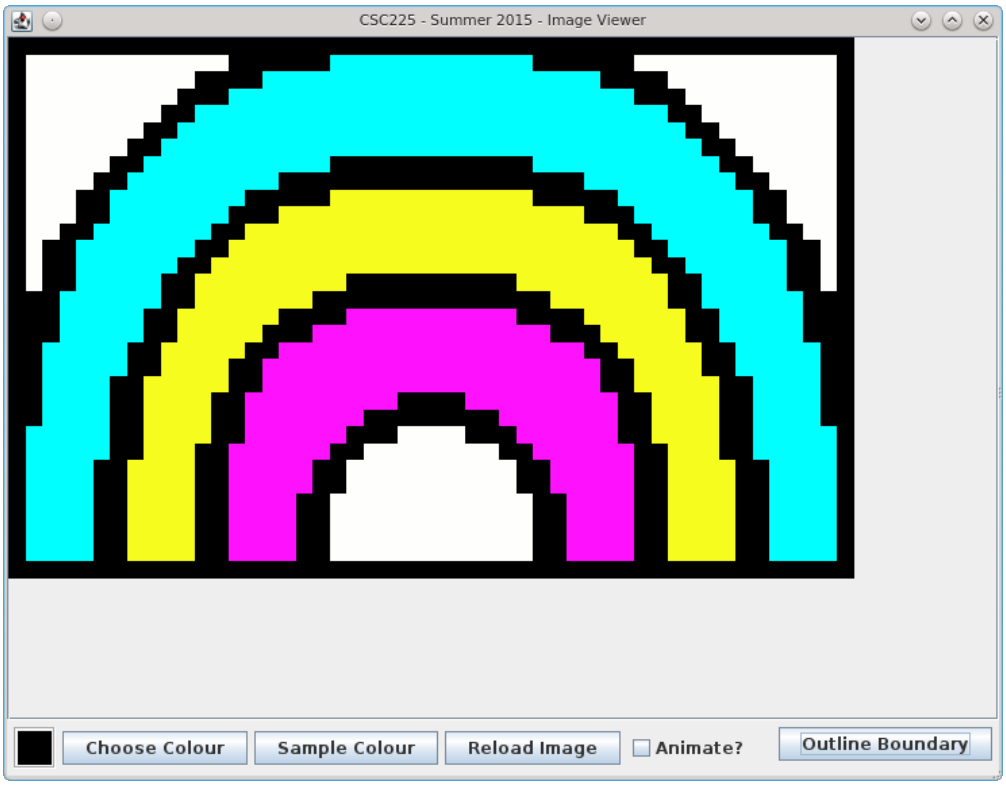


Figure 9: The image viewer window after successfully outlining the boundary of the `rainbow.png` image using black as the outline colour.

4 Test Images

A collection of test images (including the images in Figure 1) have been posted to conneX. The image viewer is capable of loading any image in PNG format, so you can also use externally-obtained images for testing.

5 Evaluation Criteria

The programming assignment will be marked out of 30 during a one-on-one demo with an instructor. During the demo, your code will be tested and inspected, and you may be asked to explain or justify decisions you made during the implementation process. An electronic signup sheet for demo times will be posted on conneX. If you do not sign up for a demo time, or if you sign up but do not attend your demo (without receiving an exception), you will receive a zero on the assignment.

You must submit your assignment electronically through conneX as usual; during your demo, the only version of your code that will be evaluated is the version submitted to conneX.

To be properly tested, every submission must compile correctly as submitted, and must be based on the provided template files. **If your submission does not compile for any reason (even**

trivial mistakes like typos), or was not based on the provided template files, it will receive at most 5 out of 30. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor **before** the due date.