CSC260S: Scientific, Graphical and Symbolic Computation
Department of Computer Science
University of Toronto

Instructor: Eugene Fiume | Assignment 3 | Due Date: 7 April 1995

You will not be using Maple in this assignment. Also, as I mentioned, I will have a very generous "late" policy for this assignment to allow student to do better time management. But that doesn't mean you should wait until after classes to start! The questions will be worth about 25% for each of questions one and two, and 50% for question 3.

I will be making available an image processing facility. You will be writing that facility in Question 2, but for Question 1, use the provided software. The extension of our work to two dimensions is straightforward, but it will turn out that subscripting the filter internally will be a pain. The following applies to both question 1 and (the upcoming) question 2.

A discrete two-dimensional filter $f : \mathbf{Z}^2 \to \mathbf{R}$ is an array of real valued weights $f[i,j]$. Generally the sum the weights is one, but that is not necessary. A discrete two-dimensional image $I[i,j]$ is also an array, but this time of pixel values. To be consistent, we will assume that the elements of both filters and images are ordered in the same way: the first co-ordinate *i* refers to the *row* index and the second co-ordinate *j* refers to the *column* index. Images are numbered so that $I[0,0]$ corresponds to the bottom left corner. If there are *r* rows and *c* columns in *I*, then $I[r-1, c-1]$ is the top-right corner. Filters can be rectangular rather than just square. They may have *r* rows and *c* columns, but for convenience we will require that *r* and *c* both be *odd*. This allows a filter to have a distinct integral centre. A filter differs from an image in that its origin (0,0) is its centre. Thus, technically the elements of a filter $f[i,j]$ have the bounds $-r/2 \le i \le r/2$ and $-c/2 \le j \le c/2$. These divisions are taken to be integer divisions (e.g., 3/2 is 1). I say "technically" because in C you cannot have negative array subscripts, so C programmers will find it a bit more of a chore than Turing programmers to get their subscripts correct.

The *2-D discrete convolution sum* for any output value $(a,b) \in \mathbf{Z}^2$ is given by

$$\hat{I}[a,b] \;=\; \sum_{i=-r/2}^{r/2} \sum_{j=-c/2}^{c/2} f[i,j] I[a+i, b+j].$$

This form of convolution is a bit more convenient to use than (and is functionally equivalent to) the definition of convolution in the book.

Here are some computational notes. Image pixel values are be natural numbers in the interval [0,255]. This represents grey values in an image, where 0 is black and 255 is white. The right hand side of the convolution sum is to be thought of as a real-valued result, but when assigning this result to the left hand side one must ensure that:

- if the result is less than zero, the value assigned is zero.
- if the result is greater than 255, the value assigned is 255.
- after the computation, the assigned result is a natural number, not a real.
- indexing beyond the resolution of the image should return zero.

It is possible to take advantage of the fact that image values are "clamped" to a fixed interval.

(1) Using the provided software, design a separate filter for each of the following operations.

(a) Brightens all image pixel values by a factor of 1.2.

(b) Darkens all image pixel values by a factor of 1.2 (i.e., the inverse of (a)).

(c) Leaves regions containing lines or other sharp changes bright, but makes everything else go dark.

(d) Makes edges sharper but otherwise leaves the image intelligible.

(e) Moves the image 10 pixels down.

(f) Moves the image 10 pixels to the right.

(g) Performs blurring of each *row* of the image, but not the columns.

(h) Performs a horrible disfiguration of your scanned professor's face (elf.pgm). I may award a prize for the most horrible disfiguration.

While (a)-(f) denote fairly "atomic" filtering operations, it is actually easy to perform a sequence of operations. For example, you could blur an image and then enhance its edges.

Sample images and a couple of sample filters will be provided, as will instructions on how to use the demo.

In addition to verifying correctness, you will be evaluated on the compactness of the filter that achieves the (correct) result.

To operate the software for this question, everyone will be programming in C. But don't worry, C is only being used as a way of executing the image processing commands. You won't be programming much at all. See the posting. For Question 2, you can use either C or Turing as you wish.

(2) Using the provided software (see the postings) and the experience you gained using the package in Question 1, implement 2-D convolution and supporting software in either C or Turing. Much more information will be provided by news postings. The relevant routine is called `ApplyFilter`.

(3) In an actual design setting, some physical properties of curves or surfaces can be modelled. This can be useful for practical applications. As an example, you will be modifying your spline package from the previous assignment to assess the behaviour of curves according to two different criteria:

- *arc length*. The computation of the arc length of a parametric space curve can be found in the notes. This would warn a user that a piece of wire is being stretched too far, for example.
- *strain energy*. Imagine a curve as a metal rod under tension. Then the greater the *curvature* of the rod, the greater the tension on it. A generally accurate indicator of curvature (which itself is a messy equation) is the second derivative. We shall define the strain energy of a curve $f(t)$ over an interval $[a,b]$ as

$$S(f,[a,b]) \;=\; \int_a^b f''(t)^2\,dt.$$

Notice that this value is always positive. The strain energy of a space curve $(x(t),y(t))$ is simply

$$S(x,\,[a,b]) \;+\; S(y,\,[a,b]).$$

For a polynomial space curve, the strain energy can be computed in closed form, whereas the arc length must be numerically computed (for curves of degree greater than 2). In your OOT/C code, you should compute arc lengths using compound Simpson's rule.

Your integral should be accurate to the nearest whole number, assuming arc lengths are measured in pixel units on a screen of resolution 512×512 (nominally). You should decide on the number of subintervals over which to compute arc lengths by using your code to determine the quality of the approximation for "typical" cubic curves to be displayed on a screen. Provide a justification of your choice of the number of subintervals in the README. The interface and the action of Spline.t will be modified so that the user can choose to check the segment-wise arc length or strain energy (but not both simultaneously). If the arc length or strain energy (as chosen) of any curve segment exceeds a specified user value, then colour the curve segment red, otherwise colour it green. A new interface for Spline.t will be made available. Add a new function:

```
ChooseChecking ( t: curveCheck, r: real )
```

which will allow a user to choose a curve checking option and the threshold value to check for. The type declaration for `curveCheck` will also be provided. It will allow a user to select an arc-length check, a strain-energy check or no checking, which would return you to the same functionality as in Assignment 2.