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

Instructor: Eugene Fiume — Assignment 1 — Due Date: 10 February 1995

The objective of this assignment is to explore some aspects of parameterisation. You will be using Maple to display and define some parametric functions. You will then implement a plotting routine in Object-Oriented Turing or C.

1) This question is derived from Exercise 1.26 from the notes. Construct and plot parametric representations for: a cylinder of radius r, height h and centre (0,0,0) (see Figure 1.9), a cone with height h, apex (0,0,0), and radius r at top (Fig. 1.10), and a "flower" as in Figure 1.10 with the same circular radius r at height h. Put your code into Maple procedures **`Cylinder(r,h)`**, **`Cone(r,h)`**, **`Flower(r,h)`**, as appropriate.

2) Write a Maple procedure called **`reparam(p, [a,b], t)`** that takes a polynomial **`p`**, a domain **`[a,b]`**, and a free variable **`t`** as arguments. The routine should return a new polynomial in the free variable that is the reparameterisation of the input polynomial onto the domain [0,1]. Suppose, for example, I wish to repameterise the polynomial $x^2 - 1$ from the domain [–2,2] to [0,1]. Then your routine should act as follows:

```
> p := x^2 -1;
                                 2
                           p := x  - 1

> reparam(p, [-2,2], t);
                             2
                         16 t  - 16 t + 3
```

Make sure you simplify the polynomial you return to the user by gathering up terms. (Hints: Maple functions **`indets, sort`** and **`collect`**.

3) Using the Maple code from Figure 1.19, design a replacement rule for line segments other than the von Koch rule that results in a pretty image when the iteration described on p47 of the notes is used. You will need to understand what the **`Rotate`** command does, so try it when, for example, **`Q=[0,0]`**.

4) *Note:* don't be frightened by the notation in this question. Part of the exercise is to get you used to notation. Recall that for a line segment from (0,0) to (x,y) in the first octant, $x > 0$, the optimal line drawing algorithm defines a sequence of pixels (i,j), where row j is got by a rounding operation involving column i and the line's slope. Let $R_o(P_0,P_1)$ denote the resulting rendering using the optimal line rendering algorithm, with P_0 at the origin and $P_1 = (x,y)$ as above. In this question we shall compare this algorithm to another.

   Two points are *eight-connected* if they lie within adjacent pixels. More specifically, the corresponding x and y pixel values can each differ by no more than 1. Formally,

$$conn((x_0,y_0),(x_1,y_1)) = \begin{cases} \textbf{true} & \text{if } |\langle x_0\rangle - \langle x_1\rangle| \le 1 \textbf{ and } |\langle y_0\rangle - \langle y_1\rangle| \le 1. \\ \textbf{false} & \text{otherwise.} \end{cases}$$

   The notation $\langle x\rangle$ means the round of real or rational value x to the nearest integer. We extend the notation to points as follows: if $P = (x,y)$, then $\langle P\rangle = \big(\langle x\rangle, \langle y\rangle\big)$. Note that definition of *conn* allows a point to be eight-connected to itself. We now define an interesting recursive rendering scheme R_m based on dividing a line segment at its midpoint. Our definition is:

$$R_m(P_0,P_1) = [P_0, P_1]_m \cup \langle P_1\rangle,$$

where

$$[P_0, P_1]_m = \begin{cases} \langle P_0 \rangle & \text{if } conn(P_0, P_1) \\ \left[P_0, \frac{P_0 + P_1}{2}\right]_m \cup \left[\frac{P_0 + P_1}{2}, P_1\right]_m & \text{otherwise} \end{cases}$$

If (A,B,C) and (D,E,F) are sequences, then the *union* operation $\cup$ used above just means

$$(A,B,C) \cup (D,E,F) = (A,B,C,D,E,F).$$

Implement both R_o and R_m as Maple procedures

```
Ro := proc (x0,y0, x1, y1)
Rm := proc (x0,y0, x1, y1)
```

that return an ordered sequence of pairs that comprise the rendering:

```
[ [x0,y0], ... , [x1,y1]].
```

(a) R_o and R_m do not produce the same rendering for all endpoints in the first octant with one endpoint at the origin. Find an example for which they do not agree.

(b) Despite (a), these two renderings agree on an infinite, nontrivial class of line segments (not just, for example, all line segments lying on the x-axis). What is such a class? Use Maple to demonstrate this.

5) You will be adding to an OOT object called *Plot*, stored in "Plot.t" that will allow you to plot parametric functions in OOT. A stub for *Plot* will be provided, but two main routines must be written:

```
proc Plot2D(
    function x(t:real): real,
    function y(t:real): real,
            t0, t1: real, nt: int)

proc Plot3D(
    function x(s,t:real): real,
    function y(s,t:real): real,
    function z(s,t:real): real,
             t0, t1: real, nt: int,
             s0, s1: real, ns: int)
```

Here, **x,y,z** are functions written in Turing, **t0,t1** is the domain $[t0, t1]$ in t, **s0,s1** is the domain $[s0, s1]$ in s, and **nt**, **ns** are the number of steps in t and s respectively. That is, our step sizes are $\Delta t = (t1 - t0)/nt$ and $\Delta s = (s1 - s0)/ns$, and you will be evaluating the relevant functions at $i\Delta s$ and $j\Delta t$ for $0 \le i \le ns$ and $0 \le j \le nt$. Several sample Turing functions will be provided. Write your code so that you avoid needlessly recomputing values of **x,y,z**, and also avoid unnecessarily drawing lines that have already been drawn. In particular, you should never compute $x(s,t)$, $y(s,t)$ or $z(s,t)$ more than once for a specific s,t. When you have written your module and you are ready for a really severe test, I'll make sure I have one ready for you.

This exercise can be implemented using the C programming language if desired. Additional information regarding this will be posted.