ICFP Programming Contest 2003

Koen Claessen Nils Anders Danielsson Niklas Eén
Daniel Hedin John Hughes Ulf Norell
Karol Ostrovsky Josef Svenningsson Tapani Utriainen

June 29, 2003

1 Introduction

The object of this year's programming contest is to drive a simulated car around a set of racing tracks as fast as possible. The solutions you submit will consist of a trace of the commands you give to the car at each moment of the simulation. We will run your commands on our simulator to determine your finishing time, and the team with the best overall performance will be crowned the winners!

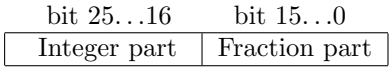
Since your submission is just a set of traces, we will not need to *run* your program on our computers, which means you may use any programming language for which *you* have an implementation. You may also, if you wish, write an interactive program and "drive the car" yourself to construct the trace, although it is unlikely this will beat a calculated or post-optimized trace. If you have a super-computer available, you may use it. If you can harness a thousand workstations to search for the fastest trace, good for you! You can even write the trace by hand (although we don't recommend it!).

We still require you to send in the source code you write for the contest. We also invite you to send us tracks of your own design, which we will publish on the web after the contest. Either of these may influence the award of the "Judges' Prize". We also encourage you to publish a web page yourselves about your entry after the competition is over.

Let the games begin!

2 Fixed Point Arithmetic

All calculations in the simulation take place using integers with a 16-bit fractional part (in $1/65536^{th}s$ if you wish). You may assume the integer part to be at most 10 bits:



Let "/" to denote integer division, which is always rounded towards zero:

$$\begin{aligned} 8/3 = (\text{-}8)/(\text{-}3) &= 2 \\ (\text{-}8)/3 = 8/(\text{-}3) &= \text{-}2 \end{aligned}$$

Furthermore, let << and >> denote the left and right bit-shift operations respectively. Addition, subtraction, negation, and inequality operators on fixed point numbers are implemented by the normal integer operations. Multiplication and division on fixed point numbers x and y are defined as follows:

mul(x, y) { **return** (x * y) >> 16 }
div (x, y) { **return** (x << 16) / y }

Note that 32 bits is not enough for implementing these operations. We also define *sine* and *cosine* over the interval $-\pi \leq x \leq \pi$:

```
sin(x) {
    if (x < 0)          return -sin(-x)
    if (x > (π/2)fix)   return sin(πfix − x)
    x2 = mul(x, x)
    x3 = mul(x, x2)
    x5 = mul(x3, x2)
    x7 = mul(x5, x2)
    return x − x3/6 + x5/120 − x7/5040
}
cos(x) {
    x += (π/2)fix
    if (x > πfix) x −= (2π)fix
    return sin(x)
}
```

where:

$$\begin{aligned} (\pi/2)_{fix} &= 102944 \\ \pi_{fix} &= 205887 \\ (2\pi)_{fix} &= 411775 \end{aligned}$$

3 Car Simulation Model

The state of the car is described by four variables:

- x – The x-coordinate of the position of the car.
- y – The y-coordinate of the position of the car.
- v – The velocity of the car: $v \geq 0$.
- d – The direction of the car: $-\pi \leq d \leq \pi$.

The simulation will proceed in time-steps. At each step the car can *accelerate*, *turn left*, *turn right*, or *brake*. It is illegal to turn left and right simultaneously. It is also illegal while braking to perform any other action (you cannot turn or accelerate when braking). You may, however, accelerate and turn at the same time, or just let the car roll (do nothing), for a total of seven possible maneuvers at each time-step. The simulation is parameterized by the following constants (provided in fixed point arithmetic):

A	$= 24$	– Acceleration factor.
B	$= 36$	– Braking factor.
T	$= 64$	– Turning factor.
L	$= 20000$	– Limit on turn.
$F_{0,1,2}$	$= 4, 12, 24$	– Friction and air-resistance.

Using these, the state variables are updated from one time-step to the next as follows:

v $-=$ F_0 + *mul*(F_1, v) + *mul*(F_2, *mul*(v,v)))
if ("accelerate") v $+=$ A
if ("brake") v $-=$ B
if (v $<$ 0) v $=$ 0

if ("turn left") d $-=$ *div*(T, *mul*(v,v) + L)
if ("turn right") d $+=$ *div*(T, *mul*(v,v) + L)
while (d $< -\pi_{fix}$) d $+=$ $(2\pi)_{fix}$
while (d $> \pi_{fix}$) d $-=$ $(2\pi)_{fix}$

x $+=$ *mul*(v, *cos*(d))
y $+=$ *mul*(v, *sin*(d))

4 Track Format

Tracks are described as matrices where each element corresponds to a 1×1 square of a uniform surface, such as asphalt (the road) or grass (impassable). Note that in the fixed point arithmetic a square is thus 65536×65536 fractional units. The file format is text-based and looks as follows:

- Line 1: The *width* of the track map as an integer.
- Line 2: The *height* of the track map as an integer.
- Lines 3 to 2+*height*: The track data; each line containing *width* characters.

We give an example track together with the meaning of the characters in the data:

```
11
8
gggrwrwrggg
ggw..!..wgg
gr...!*..rg
gw..bbb..wg
gr..bbb..rg
gw.......wg
ggr.....rgg
gggwrwrwggg
```

`.`	Road (a period). Here you can drive!
`*`	Start position (considered as road).
`!`	Goal line.
letter	Impassable.

The different letters for impassable are just for nicer looks: *g=green, r=red, w=white, b=blue* (no other letters are used in our maps). Furthermore:

- All maps of the competition will be of size 1024×768. You may want to create maps of other sizes for experimental purposes (there is a tool PNG2TRK available to help you do this). We encourage you to contribute interesting maps to us!

- The data is given in growing x- and y-coordinate order. The first square of the file covers the area [0,1)×[0,1); the last [width-1,width)×[height-1,height).

- Initially, the car is positioned at the upper-left corner of the start square ("`*`"). A track may only contain one start square. The car is facing right (the angle d is 0) and has no speed (v is also 0). In the above example, the start position is (6, 2), or in fixed point arithmetic (6 * 65536, 2 * 65536).

- It is considered a crash to be located on an impassable square at any time-step. The simulation progresses discretely, so points between the current position and the previous position should *not* be considered (in particular, you can tunnel through corners at high speed). The car's top speed is guaranteed to be less than one square per time-step.

- The track is finished if the car enters the goal-line from the left. Formally: the car should be located on a goal square with a greater x value than the previous time-step.

- You cannot enter the goal-line from the right. Formally: It is considered a crash to be located on a goal-line unless the previous condition is fulfilled.

5 Trace File Format

A solution consists of a trace of commands given to the car at each time-step. It is the participants' responsibility to accurately implement the simulation model so that identical results will be produced by our implementation. We provide a limited web interface to our simulator, which, given a trace you generated for any of the tracks, simulates it and returns to you a summary of the simulation results. You may use this interface to compare your simulator to ours.

The text based format used for the trace file is a sequence of the following commands:

`.`	Roll (no acceleration, braking or turning).
`a.`	Accelerate.
`b.`	Brake.
`l.`	Roll and turn left.
`r.`	Roll and turn right.
`al.`	Accelerate and turn left.
`ar.`	Accelerate and turn right.

Each command is a set of letters terminated by a full-stop; the order of the letters is not important[1]. White-spaces are ignored. End-of-file marks the end of the trace. Example:

```
a.a.al.al.al.b.b..
```

The car accelerates the first two time-steps, then turns left while continuing to accelerate for three time-steps, then brakes for two time-steps, and finally just roll for one time-step.

[1] Revision 2: Order of letters within a command irrelevant.

6 Tie-breaker: The Rally Circuit

In the event that more than one team turns in an identical performance on the ordinary tracks, the contest will be decided by a tie-breaker — a race on a rally circuit. On the rally circuit, it *is* allowed to turn and brake at the same time — but the car may skid. Controlled skidding is the secret of success.

During a skid, the car might not face the direction it is travelling in. We let d remain the direction it is facing, and add to the state t, the direction of motion. A skid ends when d and t are close enough together. One time step becomes[2][3]:

turn = *div*(T, *mul*(v,v) + L)
diff = *anglediff*(t,d)
if (*abs*(diff) < turn **or** v == 0) t = d
v −= F_0 + *mul*(F_1, v) + *mul*(F_2,*mul*(v,v)))

if (t ≠ d) "skidding" = true
else if ("turning left or right and brake") "skidding" = true
else "skidding" = false

if ("skidding")
{ turn = *div*(2*T,L)
  if ("not accelerate") v −= B_{skid}
  if (v < 0) { v = 0; t = d; **return** }
  if ("accelerate")
  { v_x = *mul*(v,*cos*(t)) + *mul*(A_{skid},*cos*(d))
    v_y = *mul*(v,*sin*(t)) + *mul*(A_{skid},*sin*(d))
    v = *sqrt*(*mul*(v_x,v_x) + *mul*(v_y,v_y)
    t = *atan2*(v_y,v_x)
    if (*mul*(*anglediff*(t,d),diff) < 0 and *abs*(diff) < $(\pi/2)_{fix}$) t = d } }

else { **if** ("accelerate") v += A
  if ("brake") v −= B
  if (v < 0) v = 0
  turn = *div*(T, *mul*(v,v) + L) }

if ("turn left") d −= turn
if ("turn right") d += turn
while (d < $-\pi_{fix}$) d += $(2\pi)_{fix}$
while (d > π_{fix}) d −= $(2\pi)_{fix}$
if ("not skidding") t = d

x += *mul*(v, *cos*(t))
y += *mul*(v, *sin*(t))

Here $A_{skid} = 22$ and $B_{skid} = 32$ (reduced by 10%).

Rally traces begin with an `x`, followed by a sequence of trace commands of the same form as before. The new possible commands are `bl.`, `br.`, `abl.` and `abr.`.

[2]Revision 1: Third assignment to turn moved, constants A_{skid} and B_{skid} introduced.
[3]Revision 3: Conditional assignment to t added after d is updated.

7 Extended Fixed Point Arithmetic

The skidding model requires some more sophisticated fixed point arithmetic operations, which are given here[4].

```
abs(x) {  if (x < 0) return −x
          else return x }

sqrt(x) {  u = max(x,(1)fix)
           l = 0
           while (u ≠ l) {  g = div(u+l,(2)fix)
                            if (mul(g,g) ≥ x) u = g
                            else l = g + 1 }
           return u }

asin(x0) {  x = 0
            x −= div(sin(x) − x0,cos(x))
            x −= div(sin(x) − x0,cos(x))
            x −= div(sin(x) − x0,cos(x))
            return x }

atan(k) {  if (k < 0) return −atan(−k)
           if (k > (1)fix) return (π/2)fix − atan(div((1)fix,k))
           return asin(div(k,sqrt((1)fix+mul(k,k)))) }

atan2(y,x) {  if (x = 0)  if (y > 0) return (π/2)fix
                          else if (y < 0) return −(π/2)fix
                          else return 0
              else if (y = 0)  if (x > 0) return 0 else return (π)fix
              else {  flip = false
                      mirror = false
                      if (x < 0) y = −y, x = −x, flip = true
                      if (y < 0) y = −y, mirror = true
                      if (y < x) result = atan(div(y,x))
                      else result = (π/2)fix − atan(div(x,y))
                      if (flip) result −= (π)fix
                      if (mirror) result = −result
                      return result } }

anglediff(u,v) {  diff = u−v
                  if (diff < −(π)fix) diff += (2π)fix
                  else if (diff > (π)fix) diff −= (2π)fix
                  return diff }
```

[4]Revision 3: test for > replaced by ≥ in *sqrt*. This correction was also made in our implementation.

8 Submission

You will submit your solutions to us by uploading a compressed file archive to our web server, containing traces for some or all of the tracks, your source code, *etc.* Details of the submission process are explained on our web site. Tracks you design may also be submitted in a similar way.

Entries must be submitted at or before 23:59 GMT on Monday, June 30th. Entries submitted during the first 24 hours of the contest also compete in a "Lightning Division".

9 Winning

There are two categories of tracks, the ordinary ones and the rally circuit. The primary objective is to solve all the ordinary tracks in as few simulation steps as possible. In the event that several teams solve the ordinary tracks in the same total time, the solution for the rally circuit will be used to decide a winner.

The winning criteria are, most important first:

1. Number of ordinary tracks solved.
2. Sum of the simulation times for the ordinary tracks (lower is better).
3. Having a valid solution to the rally circuit.
4. The time for the rally circuit.

The "Judges Prize" will be awarded at our whim. Factors the judges may take into account include the software you enclose, a good solution to the tiebreaker track, or a beautiful track of your own submitted during the contest.