C-Sheep: Controlling Entities in a 3D Virtual World as a Tool for Computer Science Education

Eike Falk Anderson and Leigh McLoughlin

The National Centre for Computer Animation
Bournemouth University, Talbot Campus
Fern Barrow, Poole, Dorset BH12 5BB, UK

Bournemouth University

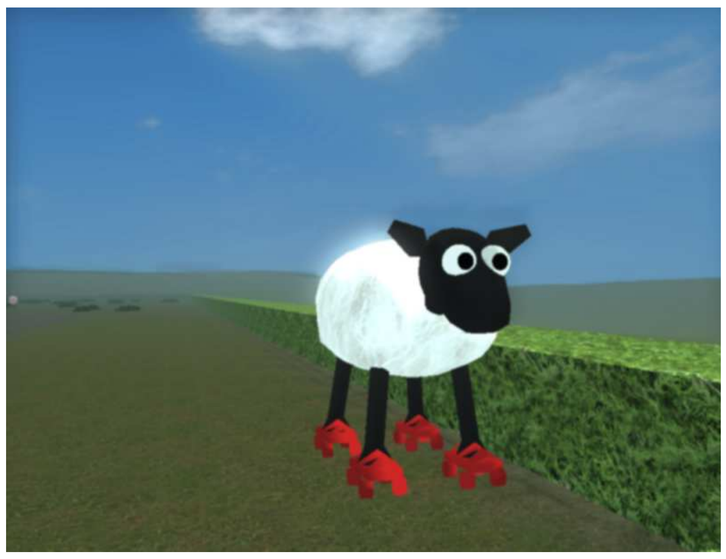
The National Centre for Computer Animation

ABSTRACT

One of the challenges in teaching computer science in general and computer programming in particular is to maintain the interest of students, who often perceive the subject as difficult and tedious. To this end, we introduce C-Sheep, a mini-language-like system for computer science education, using a state of the art rendering engine, usually found in entertainment systems. The intention is to motivate students to spend more time programming, which can be achieved by providing an enjoyable experience. In the C-Sheep system this is aided by the visual gimmickry of modern computer games, which allows programs to provide instant visualisation of algorithms. This visual feedback is invaluable to the understanding of how the algorithm works, and - if there are unintended results - how errors in the program can be debugged.

The C-Sheep System

Figure 1: Sheep entity on "The Meadow"



Introduction and Problem

Recent studies suggest that computer science education is fast approaching a crisis - enrolment numbers to degree courses have fallen to extremely low levels [4]. One of the reasons why that may be is a general misconception of computer science and programming among prospective students. A lack of knowledge of the subject area may well be the underlying reason for the bias against computer science. Computer science and related subjects are wrongly conceived as difficult and boring (non-creative) and computer programming especially is often regarded as a monotonous and uninteresting task. Computer programming is an essential skill for software developers and as such is always an integral part of every computer science curriculum. However, even if students are pursuing a computer science related degree, it can be very difficult to interest them in the act of computer programming, the writing of software, itself.

It is generally understood that programming cannot be learned from reading books on the subject but only by practicing it, yet therein lies the problem:

"how can students be motivated to practice programming?"

Rationale for Educational Programming Languages

It has been suggested that one way to achieve a greater uptake of computer science would be to make programming "more fun" [4]. Using a mini-language [3] as the introductory programming language partially fulfils this requirement. These mini-languages usually provide a task-specific set of instructions and (sensor) queries which allow users to take control of virtual entities, acting within a micro world (see figure 1). This micro world provides a graphical representation of the algorithms used in the programs controlling the virtual entities. Their position and orientation within the virtual world visualise the current state of the program which is especially useful as many problems faced by novice programmers can possibly be traced back to an inadequate understanding of program state [6].

The mini-language system "Karel the Robot" for example [9] (using a 'toy-language' based on the syntax of the Pascal programming language) is one of the best known computer science teaching tools and has had considerable success. The aim for all of these systems is to motivate students to take up programming and to provide them with an enjoyable experience at the same time. This is achieved by providing a game-like setting for the task of computer programming.

Until now, most mini-language systems used in teaching have employed a strictly 2D top-down representation, some use pseudo 3D graphics (using isometric projection) with - surprisingly - only a few examples, such as Alice [5], using true 3D graphics. Dann et al state that a 3D representation of the program state is "intrinsic in the natural way to view the data itself" [6]. It is therefore especially important to aim for a 3D game-like representation in mini-languages to interest the "Plug&Play generation" in computer programming.

C-Sheep Programming Language

The C-Sheep programming language is a subset of the ANSI C programming language [7] (see figure 2). Apart from just being a tool for learning the basics of the C programming language, C-Sheep implements the C control structures that are required for teaching the basic computer science principles encountered in structured programming. These control structures are the (unconditional) sequence, conditional statements and loops [2]. C-Sheep also allows the definition of sub-routines (functions) which can be called recursively. Unlike other teaching languages which have minimal syntax and which are variable free to provide an environment with minimal complexity, C-Sheep allows the declaration and use of variables.

To access library functions, C-Sheep programs must include header files (supposedly) containing function prototypes. This is done only to introduce novice programmers to the concept of code modularisation and libraries, while internally the C-Sheep library functions are actually intrinsic to the system. The C-Sheep standard libraries (see table 1) provide a number of general purpose functions, as well as functions for controlling sheep entities in the virtual environment. Some of the latter allow the querying of changes in the virtual world (e.g. the current state of the weather - see figure 3). These changes can be instigated interactively by the user (while programs are running).

To execute native C-Sheep programs they must be compiled with the C-Sheep compiler which translates from the C-Sheep subset of the C programming language to virtual machine bytecode.

C-Sheep Standard Libraries

stdlib.h (subset of ANSI C stdlib.h)
- abort
- exit
- rand

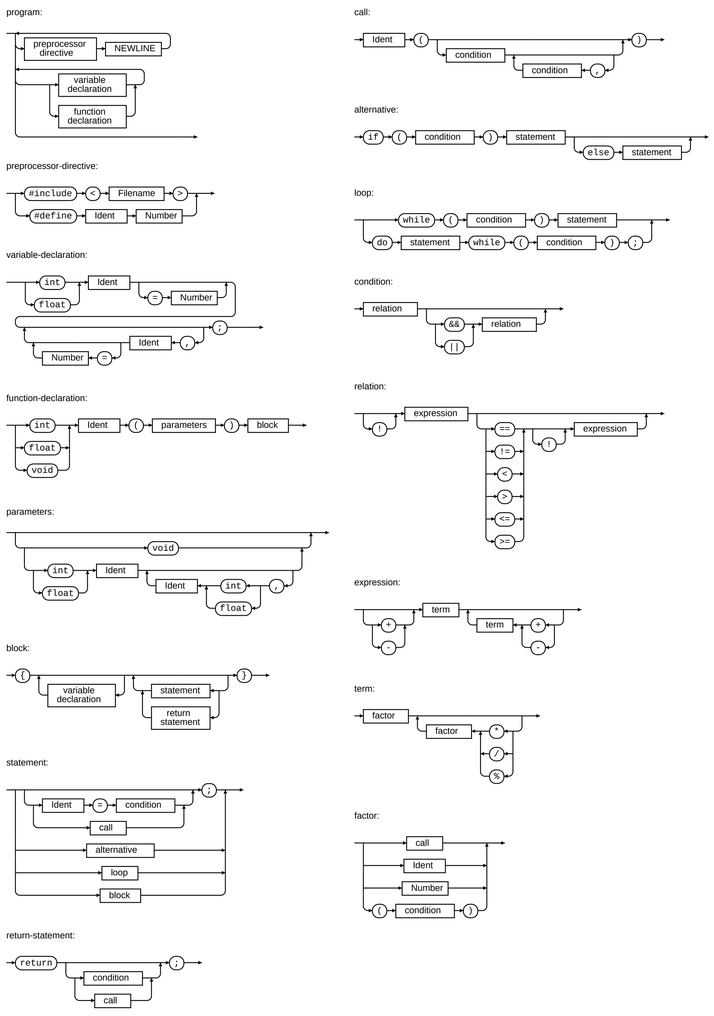
math.h (subset of ANSI C math.h)
- sqrt

sheep.h (sheep entity control functions)
- pause
- initialise
- step
- backstep
- turn_left
- turn_right
- blocked
- found
- query

Table 1: Functions of the C-Sheep standard libraries

C-Sheep Syntax

Figure 2: Syntax Diagrams for the C-Sheep Programming Language



"The Meadow" Virtual Environment

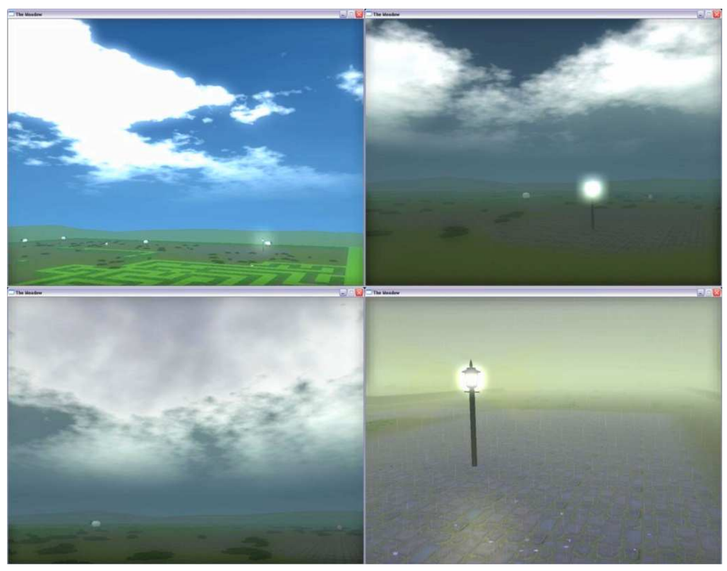
"The Meadow" virtual environment is the virtual world in which entities controlled by C-Sheep programs exist. It is based on our proprietary "Crossbow" game engine which incorporates a virtual machine for executing C-Sheep programs.

The Crossbow Engine is a compact game engine designed specifically for "The Meadow", yet it is flexible in design and offers a number of features common to more complex engines [8].

The Crossbow Virtual Machine is a module of the Crossbow Engine. It is an improvement on the ZBL/0 virtual machine [1] The instructions of the virtual machine include the use of pointers as well as facilities for the creation of aggregate data types (arrays and record structures). The virtual machine is targetable by compilers for different languages, i.e. a Java based J-Sheep or Pascal based P-Sheep could be created with relatively little effort.

Weather in "The Meadow"

Figure 3: "The Meadow" is alive with the sound of sheep



Lua Cg SDL Simple Directmedia Layer

C-Sheep C Library

As recommended by Untch [10], C-Sheep provides a counterpart library for C, mirroring the C-Sheep library functions of the virtual machine. This allows C-Sheep programs to be compiled into an executable using a normal off-the-shelf C/C++ compiler. This executable can then be run from within the native working environment of the operating system. The purpose of this library is to simplify the migration from the educational mini-language to real-world systems by allowing novice programmers to make an easy transition from using the C-Sheep system to using the C programming language.

Sample C-Sheep Program

```
#include <sheep.h>

#define FALSE 0
#define TRUE  1

void turn_round(void)
{
    turn_left();
    turn_left();
}

void go(void)
{
    if(found(BALL)==FALSE)
    {
        if(blocked(RIGHT)==TRUE)
        {
            if(blocked(FRONT)==FALSE)
            {
                step();
                go();
                step();
            }
            else
            {
                turn_left();
                go();
                turn_right();
            }
        }
        else
        {
            turn_right();
            step();
            go();
            step();
            turn_left();
        }
    }
    else
    {
        turn_round();
    }
}

int main(void)
{
    initialise(43,11); /* start at entrance to maze */
    go();
    /* end the program when the original position has been reached */
    turn_round();
    return 0;
}
```

A sheep running this program will traverse a maze, looking for a ball. Once a ball is found, the sheep will stop looking and retrace its steps until it reaches its starting point (achieved by recursion). The strategy used for traversing the maze is to always follow the right wall.

Summary and Future Work

We have presented C-Sheep, a system for the teaching of computer science principles using a subset of the C programming language within a 3D computer game-like virtual environment. We believe that our system is suitable for the task it was designed for, but conclusive proof will only be available after we have collected data from a trial run of the C-sheep system. For this we are currently planning to introduce the C-Sheep system as a teaching tool for the first term of the first year computer programming unit at the National Centre for Computer Animation (Bournemouth University).

Acknowledgements

First and foremost we would like to express our gratitude towards our supervisor, Prof. Peter Comninos. Without his support this project would not have been possible. We would also like to thank our colleagues, especially Olusola Aina, for their comments and suggestions that have contributed to this project. Finally we need to mention Dominic Halford. It is his "fault" that our programs control sheep instead of other animals.

Lua Logo design by Alexandre Nakonechnyj (nako@openlink.com.br).

No Sheep were harmed during the creation of this poster.

References

[1] E. F. Anderson. A npc behaviour definition system for use by programmers and designers. In *Proceedings of CGAIDE 2004*, pages 203–207, 2004.

[2] C. Böhm and G. Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Communications of the ACM*, 9(5):366–371, 1966.

[3] P. Brusilovsky, E. Calabrese, J. Hvorecky, A. Kouchnirenko, and P. Miller. Mini-languages: A way to learn programming principles. *Education and Information Technologies*, 2(1):65–83, 1997.

[4] L. Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. *ACM SIGCSE Bulletin*, 38(1):27–31, 2006.

[5] S. Cooper, W. Dann, and R. Pausch. Alice: A 3-d tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, 15(5):107–116, 2000.

[6] W. Dann, S. Cooper, and R. Pausch. Making the connection: Programming with animated small world. In *Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, pages 41–44, 2000.

[7] B. W. Kerninghan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, 1988.

[8] L. McLoughlin and E. F. Anderson. I see sheep: A practical application of game rendering techniques for computer science education. In *Poster at Future Play*, 2006.

[9] R. E. Pattis. *Karel the Robot, a Gentle Introduction to the Art of Programming*. John Wiley and Sons, 1981.

[10] R. H. Untch. Teaching programming using the karel the robot paradigm realized with a conventional language. On-line at: http://www.mtsu.edu/~untch/karel/karel90.pdf, 1990.

URL: http://ncca.bournemouth.ac.uk/eanderson/C-Sheep/