ToyLOGO

CS475 / CS675: Computer Graphics - Assignment 1

Due Date: 22/8/2014

1 Introduction

Turtle graphics is a style of computer drawing where the drawing cursor is imagined to be turtle, who carries a pen. The turtle has a preserved state (position and orientation) and can perform a small number of operations against that state (move forward, move back. turn left, turn right). While these operations are performed the pen is used to plot 2D drawings on a drawing canvas (which is essentially a part of the 2D plane).

The turtle thus consists of:

1. A position
2. An orientation
3. A pen, itself having attributes such as color.

The turtle moves with commands that are relative to its own current position, such as "move forward 10 units" and "turn left 90 degrees". The turtle can also move without drawing, as if the pen is held "up." It is possible for a programmer (artist?) to understand (predict and reason about) the turtle's motion by imagining what they would do if they were the turtle. Seymour Papert (inventor of the LOGO programming language) called this "body syntonic" reasoning.

Turtle graphics was the model used in the very popular LOGO programming language. In this assignment you will implement an interpreter for toy version of the LOGO language called ToyLOGO and use that to create interesting 2D drawings and animations.

2 ToyLOGO Definition

The turtle can move forward and backward, turn left or right. The current turtle position can also be set without drawing.

We define the ToyLOGO command set that the interpreter should understand as follows.

1. Begin a ToyLOGO program.
   Syntax: **BEGIN**

2. End a ToyLOGO program.
   Syntax: **END**

3. Reset turtle position to origin, **O(0,0)** and orientation to **0** degrees, which is due east.
   Syntax: **RESET**

4. Clear the screen.
   Syntax: **CLS**

5. Move the turtle in the forward direction by *dist* units and draw a line as the turtle moves.
   Syntax: **F** *dist*

6. Move the turtle in the backward direction by *dist* units and draw a line as the turtle moves.
   Syntax: **B** *dist*

7. Turn the turtle left by *angle* degrees.
   Syntax: **L** *angle*

8. Turn the turtle right by *angle* degrees.
   Syntax: **R** *angle*

9. Move the turtle in the forward direction by *dist* units without drawing.
   Syntax: **MF** *dist*

10. Move the turtle in the backward direction by *dist* units without drawing.
    Syntax: **MB** *dist*

11. Change the current drawing color to *(*r, g, b). Red, Green and Blue values range from 0.0 to 1.0.
    Syntax: **COL** *r g b*

12. Change the current background color to *(*r, g, b). Red, Green and Blue values range from 0.0 to 1.0.
    Syntax: **BGCOL** *r g b*

13. Repeat a set of command *n* times.
    Syntax: **REPEAT** *times*

14. End of a repeat block. All commands between **REPEAT** and **ENDREP** are repeated. Nested REPEATs are allowed.
    Syntax: **ENDREP**

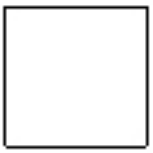
15. Scale the drawing canvas uniformly by a factor of s. The original coordinate system on the drawing canvas ranges from $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$. The origin is at the center of the canvas. This is at the default scale 1.0. This means that all values given to **F**, **B**, **MF** and **MB** will lie inside the canvas if they are in this range. Scaling can help increase or decrease this range. Note that this scaling is independent of window size.
Syntax: **SCALE** s

3 ToyLOGO Examples

1. Example 1: Here are two ways to write a program to generate a square.

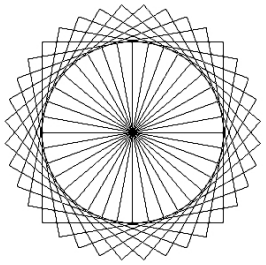
```
BEGIN
CLS
RESET
F 0.2
R 90
F 0.2
R 90
F 0.2
R 90
F 0.2
R 90
END
```

```
BEGIN
CLS
RESET
REPEAT 4
F 0.2
R 90
ENDREP
END
```



2. Example 2: This program rotates a square about it's center.

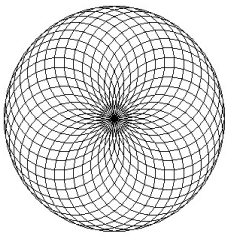
```
BEGIN
CLS
RESET
REPEAT 36
REPEAT 4
F 0.4
R 90
ENDREP
R 10
ENDREP
END
```



3. Example 3: These two programs produce output that looks the same.

```
BEGIN
CLS
RESET
REPEAT 36
REPEAT 36
F 0.05
R 10
ENDREP
R 10
ENDREP
END
```

```
BEGIN
CLS
RESET
SCALE 100
REPEAT 36
REPEAT 36
F 5
R 10
ENDREP
R 10
ENDREP
END
```



4 The Assignment

Given:

A base code for the assignment is given. It contains the following:

- A rudimentary parser for the ToyLOGO language is given (`parser.hpp / parser.cpp`. This will read a ToyLOGO program from a .logo file and convert it to an internal list of commands.
- A skeleton OpenGL application source code that will execute the commands to generate the drawings (`toylogo.cpp`).

  The Makefile can be called to compile the application program.
- A default drawing (a Koch Snowflake) that gets drawn if no filename is specified (`render_drawing.hpp/render_drawing.cpp`)
- A precompiled executable (`toylogo_complete`) is included with the base code that can be run to check and compare the output that your final program should generate.

To do:

1. A skeletal turtle class (*turtle_t*)is given. You have to implement the various functions in this class. Note that, the interpreter has already been implemented. The parser converts the ToyLOGO program to a list of commands. The application goes through this list and executes every command, using the `exec(..)` function in the *turtle_t* class.

   In order to do this you have to modify the `turtle.cpp` file. The assignment can be completed without any change to the class signature given (i.e., only the function bodies have to be filled out).

2. As a check, the example programs should produce the indicated output.

3. Next, generate at least one complex, *original* interesting looking drawing.

4. Create a HTML page that contains the ToyLEGO program and screenshots of output it produces, as generated by your submission.

5. The assignment will be marked as follows:

   - Implementing **F**, **B**, **MB**, **MF**, **R**, **L** correctly : 30 marks
   - Implementing **CLS**, **RESET**, **COL**, **BGCOL** correctly : 10 marks
   - Implementing **SCALE** correctly : 10 marks
   - Implementing **REPEAT** correctly : 15 marks
   - Original drawing : 15 marks
   - Report + Viva during Demo : 20 marks

   - Total : 100 marks
   - Late submission will follow a policy of graceful degradation with a 25% penalty for each day's delay (i.e., zero marks if the assignment is more than three days late after the due date.)
   - It is **ok** to submit a partially done assignment if you cannot complete it. You will get partial credits.
   - Please do not plagiarize source code.

6. Bonus items: **Bonus Items will be graded only if the regular assignment is complete and all correct**. So please attempt these **only after** you have finished the regular assignment.

   - Implementing a **PAUSE** command for ToyLOGO and using it to generate an animation. For e.g., the following program would generate an animation of a rotating square:

```
BEGIN
CLS
RESET
REPEAT 36
REPEAT 4
F 0.2
R 90
```

```
ENDREP
CLS
R 10
PAUSE 10
ENDREP
```

- In our implementation of the ToyLOGO language there are no variables and recursion. However, if these two features are added turtle graphics can be used to draw beautiful fractals. One such example is the *Koch Snowflake* that is drawn by default if no .logo filename is given as input. Modify `render_drawing.hpp` / `render_drawing.cpp` and replace the Koch Snowflake by another drawing of your choice that uses variables and/or recursion. Some such examples are:
  - Various kinds of spirals
  - Lindenmayer systems
  - Sierpinski triangle

To Submit:

1. A Tar-Gzipped archive of the complete source code (and only source code). It should compile using the given Makefile on any Ubuntu system.
2. Include a README file in the .tgz archive containing a link to a html report page on the assignment that should contain some details about what you implemented and images of the results that you generated.
3. The exact mode of submission will be announced later in the lecture.