eGrapher: A Programming Language for Art

Long Long: ll3078@columbia.edu
Xinli Jia: xj2191@columbia.edu
Jiefu Ying: jy2799@columbia.edu
Linnan Wang: lw2645@columbia.edu
Darren Chen: dsc2155@columbia.edu

September 28, 2016

Contents

1 Introduction and Motivation

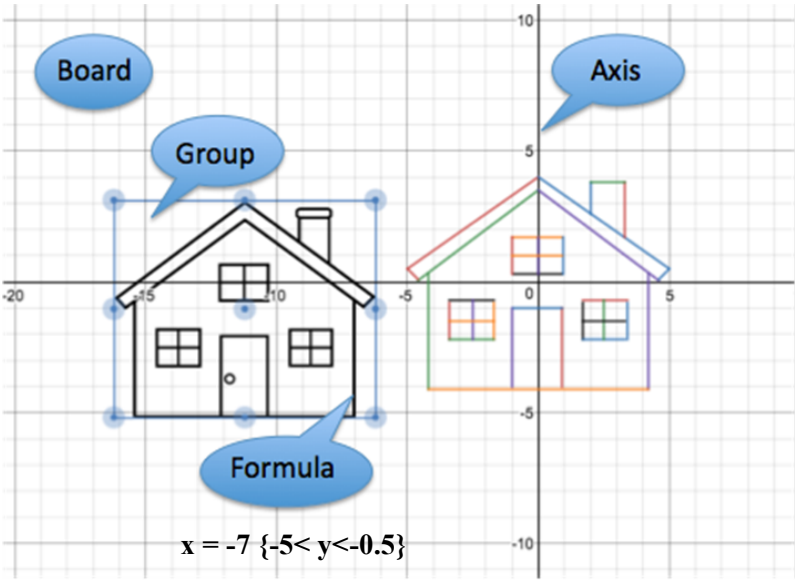
eGrapher offers an innovative way of drawing art with pinpoint accuracy. The language gives the user a wide set of tools to accurately illustrate detailed drawings compared to simple and rigid pre-installed tools such as paint on Windows. The goal of eGrapher is to make drawing on digital systems easier as well as allow for users to draw more complicated paintings through mathematical functions. The project will allow us to also better understand the intersection of mathematics and art. Users will be able to use simple syntax and math expression to draw graphs, diagrams, and more complicated objects through simple objects.

In addition, eGrapher could serve as a useful introductory language for elementary school kids combining programming, mathematics, and art into one package. Students will be able to learn how to use simple functions and loops to draw an actual picture. For example, one exercise could ask students to draw a heart through polar coordinates.

2 Language Design and Syntax

The typing style, flow control, and function statements in eGrapher takes is reminiscent of C++, while the overall design takes several interesting ideas from Matlab. It intends to throw away complex object oriented features and provide stronger build-in data type and functions to help users finish their plotting efficiently. The eGrapher compiler compiles code into LLVM and links to C library for further executions.

The eGrapher system open a main window to show the picture which users drawn using eGrapher code, while it also supports standard output stream connected to the terminal or console. In the main window, there is a big backboard. Users code to generate curves and paste them on the board then comes a graph. In order to figure out the correct place to "paste", there should be an only axis system on the board. Formula may have strong mutual relations to each other, so eGrapher offers group structure to manage formula. Board, axis, group and formula are four basic concepts form the whole plotting system of eGrapher. Take the following picture as an example, grids and axis are fixed on the white board. Two groups of formula, particularly two houses are plotted.



Since the design goal of eGrapher is to draw paintings through mathematical functions, focusing on implantation of four basic data type (board, axis, group, formula) and their build-in functions can greatly improve and guarantee plotting and coding experience.

2.1 Comments

Syntax	Comment Style
# *code*	Single line
/* *code* */	Multiple lines

Table 1: Comment Styles

2.2 Built-in Data Types

- eGrapher is statically strongly typed language
- eGrapher comes with four basic types: int, float, bool and string. Each of these basic types can be used as raw values with no prior declaration of variables or can be assigned as the values of variables.
- eGrapher also provides two built-in data types: dot and formula, which provide the basis for drawing element.
- The built-in collections are list, group and board
- In addition to these data types, eGrapher also includes the value null, which represents the absence of value for any data type.

Data Type	Explanation
int	32-bit integer
float	32-bit floating-point number
bool	Boolean value
string	Sequence of characters closed with double quotes
formula	To define function
dot	A dot on graph on (x, y) axis position
list	Sequence of numeric, Boolean, or other types
group	Combination of one or more formula and dot
axis	Set axis limits and aspect ratios
board	All the element on displayed. Contain groups and axis.

Table 2: Built-in Data Types

2.3 Operators

Category	Data type	Operator	Explanation
Arithmetic	*int, float*	+, -, *, /, %, ^	Arithmetic computation operations in the same way as mainstream languages such as C/C++ and Matlab. ^ symbol means power.
Comparison	*int, float, string*	==, ! =, <, >, <=, >=	Comparison operations supporting numbers and strings. While string equality compares the characters contained in the string.
Concatenation	*string*	+	Concatenate two strings.
Assignment	*int, float*	=, +=, −=, ∗=, /=	Assign value to variables, mostly after certain calculation.

Table 3: Operators

2.4 Data Type Built-in Functions

Category	Syntax	Explanation
Output functions	int.print (*int* i)	Print out given integer i, similar to stdout.
	float.print (*float* f)	Print out given floating number f.
	string.print (*string* s)	Print out given string s.
	display_group()	Show a particular group of formula on screen in main window.
	export_board(string filename, string type)	Export the present board to an image file. Filename and type can be specified.

Table 4: Basic data type build-in functions

Category	Syntax	Explanation
Formula management	set_expression(*string* e)	Set numerical expression for each formula. In default case, a new-built formula takes “” as its expression and empty formula will be ignored when plotting.
	set_limits({xlim}{ylim})	Both x limits and y limts can be set here. Although argument and dependent variable roles can switch, we suppose the first brace describe x limit, the second one indicates y limits.
	set_color(r,g,b)	Take three integers as rgb value of the line color.
	set_linesyle(style, width)	Both style and witdth are integers. Width simply implies line width, while adjusting style can produce dotted line, * line, things like that. A detailed map of styles and index numbers will be released later.

Table 5: Formula type build-in functions

Category	Syntax	Explanation
Dot Management	set_position(*float* x, *float* y)	Set position for a dot.
	set_color(r,g,b)	Take three integers as RGB value of the dot color.
	set_dotsyle(style, size)	Both style and size are integers. Size simply implies mark size, while adjusting style can produce dotted, *, +, marks like that. A detailed map of style and index numbers will be released later.

Table 6: Dot type build-in functions

Category	Syntax	Explanation
Group Management	add_element(f)	Add elements to a certain group. Group is quite similar to list with two data types (formula and dot in it) in it.
	delete_element(*int* index)	Delete a certain element in group using its index. Index are maintained according to the attendance order.
	set_uniform_color(r,g,b)	Paint all the elements in the group with same color. The default value of uniform color vector is (-1,-1,-1), which means no requirements.
	set_uniform_linesyle(style, width)	Similar to uniform color, this time a uniform line style applies to each formula.
	set_uniform_dotsyle(style, width)	Similar to uniform line style, this time a uniform dot style applies to each dot.
	size()	return number of elements in group
	list_formula()	return all the formula in group as a list.
	list_dot()	return all the dots in group as a list.
	set_diplay(bool)	Plot the group or not.

Table 7: Group type build-in functions

Category	Syntax	Explanation
Axis management	set_xlim(*float* x, *float* y)	Set the display limits of x-axis.
	show_xlabel(*bool*)	Whether show lables on x-axis.
	show_xaxis(*bool*)_	Whther show x-axis.
	set_ylim(*float* x, *float* y)	Set the display limits of y-axis.
	show_ylabel(*bool*)	Whether show lables on y-axis.
	show_yaxis(*bool*)_	Whther show y-axis.

Table 8: Axis type build-in functions

Category	Syntax	Explanation
Board management	show()	Open the main window and show the picture.
	List_group()	Return a list of groups.
	set_backgroud_color(r,g,b)	Set background color according to rgb value.

Table 9: Board type build-in functions

2.5 Control Flow

Syntax	Examples
while condition { /* *code* */}	***while*** (a > 1) { a -= 1 }
for init; cond; inc/dec { /* *code* */}	***for*** (i; i<10;i+=1) { ***int.print*** (i) }
for variable ***in*** list { /* *code* */}	***for*** i ***in*** L { ***int.print*** (i) }

Table 10: Loops

Syntax	Examples
if condition statements	***if*** (a==b) { a += b }
if condition statements ***else*** statements	***if*** (a==b) { a += b } ***else*** { a -=b }
if condition statements ***else if*** condition statements ***else*** statements	***if*** (a==b) { a += b } ***else if*** (a>b) { a -=b } ***else*** { a *=b }

Table 11: Condition Statements

2.6 Function Statement

Syntax	Example
fun function_name (variable1, variable2 …) { function statements }	***fun*** addition(int a, int b) { ***return*** a+b }

3 Sample Code

```
formula f1, f2, f3, f4;
f1.set_expression("(t+tcost,t+sint)");
f1.set_limit("0<=t<=10");
f1.set_color(117,103,174);
f2.set_expression("(t-tcost,t-sint)");
f2.set_limit("0<=t<=10");
f2.set_color(255,163,104);
f3.set_expression("(x-10)^2+(y-10)^2=10^2");
f3.set_color(194,107,115);
f4.set_expression("y>x");
f4.set_limit("0<x", "0<y<20");
f4.set_color(88,143,192);

dot d1;
d1.set_position(17.071,17.071);
d1.set_color(88,143,192);

group g;
g.add_element (f1);
g.add_element (f2);
g.add_element (f3);
g.add_element (d3);
Axis.xrange(0, 20);
Board.plot();
```